MỤC LỤC

LỜI NÓI ĐẦU

Ngôn ngữ lập trình java ra đời và được các nhà nghiên cứu của Công ty Sun Microsystem giới thiệu vào năm 1995. Sau khi ra đời không lâu, ngôn ngữ lập trình này đã được sử dụng rộng rãi và phổ biến đối với các lập trình viên chuyên nghiệp cũng như các nhà phát triển phần mềm. Gần đây ngôn ngữ lập trình, công nghệ java đã được đưa vào giảng dạy ở các cơ sở đào tạo lập trình viên chuyên nghiệp. Một số trường đại học ở Việt Nam dạy môn lập trình java như một chuyên đề tự chọn cho các sinh viên công nghệ thông tin giai đoạn chuyên ngành.

Sau một thời gian tìm hiểu, làm việc và được tham gia giảng dạy chuyên đề lập trình java cho lớp cử nhân tin học từ xa qua mạng. Nhóm tác giả chúng tôi quyết định biên soạn cuốn giáo trình này nhằm phục vụ công tác giảng dạy cũng như học tập của sinh viên chuyên ngành công nghệ thông tin.

Nội dung giáo trình tập trung vào những kiến thức căn bản nhất của lập trình java giúp người đọc bước đầu tiếp cập dễ dàng với công nghệ mới này, và đây cũng chính là một bước đệm để chúng ta trở thành “java shooter”. Một số vấn đề nâng trong ngôn ngữ lập trình java như: javabean, thiết kết giao diện dùng thư viện JFC(Java Foundation Class), lập trình mạng, lập trình cơ sở dữ liệu bằng java, lập trình ứng dụng web dùng J2EE (Java 2 Enterprise Edition), … sẽ được nói đến trong các chuyên đề nâng cao. Chương 6 của giáo trình giới thiệu tổng quan về lập trình cơ sở dữ liệu dùng jdbc, một nội dung theo chúng tôi cần phải được trình bày trong một chuyên đề riêng.

Để có thể đọc hiểu giáo trình này người đọc cần nắm vững các kiến thức về: nhập môn lập trình, lập trình hướng đối tượng. Đây là lần xuất bản đầu tiên chắc chắn không thể tránh khỏi những sai sót. Nhóm tác giả rất mong nhận được những ý kiến đóng góp của quý thầy cô, các đồng nghiệp và bạn đọc để có

thể hoàn thiện hơn giáo trình này phục vụ cho việc học tập của sinh viên.

Xin chân thành cảm ơn!

TPHCM tháng 01/2006
Nhóm tác giả

Chương 1: GIỚI THIỆU TỔNG QUAN VỀ NGÔN NGỮ LẬP TRÌNH JAVA

1.1.Mở đầu

Chương này sẽ cung cấp cho sinh viên các khái niệm, kiến thức cơ bản liên quan đến việc lập trình ứng dụng bằng ngôn ngữ Java như: lịch sử phát triển của java, các đặc điểm của java, khái niệm máy ảo, cấu trúc của một chương trình đơn giản viết bằng Java cũng như cách xây dựng, dịch và thực thi một chương trình Java.

1.2.Giới thiệu về ngôn ngữ lập trình Java

1.2.1. Java là gì?

Java là ngôn ngữ lập trình hướng đối tượng (tựa C++) do Sun Microsystem đưa ra vào giữa thập niên 90.

Chương trình viết bằng ngôn ngữ lập trình java có thể chạy trên bất kỳ hệ thống nào có cài máy ảo java (Java Virtual Machine).

1.2.2.Lịch sử phát triển của ngôn ngữ lập trình Java

Ngôn ngữ lập trình Java do James Gosling và các công sự của Công ty Sun Microsystem phát triển.

Đầu thập niên 90, Sun Microsystem tập hợp các nhà nghiên cứu thành lập nên nhóm đặt tên là Green Team. Nhóm Green Team có trách nhiệm xây dựng công nghệ mới cho ngành điện tử tiêu dùng. Để giải quyết vấn đề này nhóm nghiên cứu phát triển đã xây dựng một ngôn ngữ lập trình mới đặt tên là Oak tương tự như C++ nhưng loại bỏ một số tính năng nguy hiểm của C++ và có khả năng chạy trên nhiều nền phần cứng khác nhau. Cùng lúc đó world wide web bắt đầu phát triển và Sun đã thấy được tiềm năng của ngôn ngữ Oak nên đã đầu tư cải tiến

và phát triển. Sau đó không lâu ngôn ngữ mới với tên gọi là Java ra đời và được giới thiệu năm 1995.

Java là tên gọi của một hòn đảo ở Indonexia, Đây là nơi nhóm nghiên cứu phát triển đã chọn để đặt tên cho ngôn ngữ lập trình Java trong một chuyến đi tham quan và làm việc trên hòn đảo này. Hòn đảo Java này là nơi rất nổi tiếng với nhiều khu vườn trồng cafe, đó chính là lý do chúng ta thường thấy biểu tượng ly café trong nhiều sản phẩm phần mềm, công cụ lập trình Java của Sun cũng như một số hãng phần mềm khác đưa ra.

1.2.3.Một số đặc điểm nổi bậc của ngôn ngữ lập trình Java

Máy ảo Java (JVM - Java Virtual Machine)

Tất cả các chương trình muốn thực thi được thì phải được biên dịch ra mã máy. Mã máy của từng kiến trúc CPU của mỗi máy tính là khác nhau (tập lệnh mã máy của CPU Intel, CPU Solarix, CPU Macintosh … là khác nhau), vì vậy trước đây một chương trình sau khi được biên dịch xong chỉ có thể chạy được trên một kiến trúc CPU cụ thể nào đó. Đối với CPU Intel chúng ta có thể chạy các hệ điều hành như Microsoft Windows, Unix, Linux, OS/2, … Chương trình thực thi được trên Windows được biên dịch dưới dạng file có đuôi .EXE còn trên Linux thì được biên dịch dưới dạng file có đuôi .ELF, vì vậy trước đây một chương trình chạy được trên Windows muốn chạy được trên hệ điều hành khác như Linux chẳng hạn thì phải chỉnh sửa và biên dịch lại. Ngôn ngữ lập trình Java ra đời, nhờ vào máy ảo Java mà khó khăn nêu trên đã được khắc phục. Một chương trình viết bằng ngôn ngữ lập trình Java sẽ được biên dịch ra mã của máy ảo java (mã java bytecode). Sau đó máy ảo Java chịu trách nhiệm chuyển mã java bytecode thành mã máy tương ứng. Sun Microsystem chịu trách nhiệm phát triển các máy ảo Java chạy trên các hệ điều hành trên các kiến trúc CPU khác nhau.

Thông dịch:

Java là một ngôn ngữ lập trình vừa biên dịch vừa thông dịch. Chương trình nguồn viết bằng ngôn ngữ lập trình Java có đuôi *.java đầu tiên được biên dịch thành tập tin có đuôi *.class và sau đó sẽ được trình thông dịch thông dịch thành mã máy.

Độc lập nền:

Một chương trình viết bằng ngôn ngữ Java có thể chạy trên nhiều máy tính có hệ điều hành khác nhau (Windows, Unix, Linux, …) miễn sao ở đó có cài đặt máy ảo java (Java Virtual Machine). Viết một lần chạy mọi nơi (write once run anywhere).

Hướng đối tượng:

Hướng đối tượng trong Java tương tự như C++ nhưng Java là một ngôn ngữ lập trình hướng đối tượng hoàn toàn. Tất cả mọi thứ đề cập đến trong Java đều liên quan đến các đối tượng được định nghĩa trước, thậm chí hàm chính của một chương trình viết bằng Java (đó là hàm main) cũng phải đặt bên trong một lớp. Hướng đối tượng trong Java không có tính đa kế thừa (multi inheritance) như trong C++ mà thay vào đó Java đưa ra khái niệm interface để hỗ trợ tính đa kế thừa. Vấn đề này sẽ được bàn chi tiết trong chương 3.

Đa nhiệm - đa luồng (MultiTasking - Multithreading):

Java hỗ trợ lập trình đa nhiệm, đa luồng cho phép nhiều tiến trình, tiểu trình có thể chạy song song cùng một thời điểm và tương tác với nhau.

Khả chuyển (portable):

Chương trình ứng dụng viết bằng ngôn ngữ Java chỉ cần chạy được trên máy ảo Java là có thể chạy được trên bất kỳ máy tính, hệ điều hành nào có máy ảo Java. “Viết một lần, chạy mọi nơi” (Write Once, Run Anywhere).

Hỗ trợ mạnh cho việc phát triển ứng dụng:

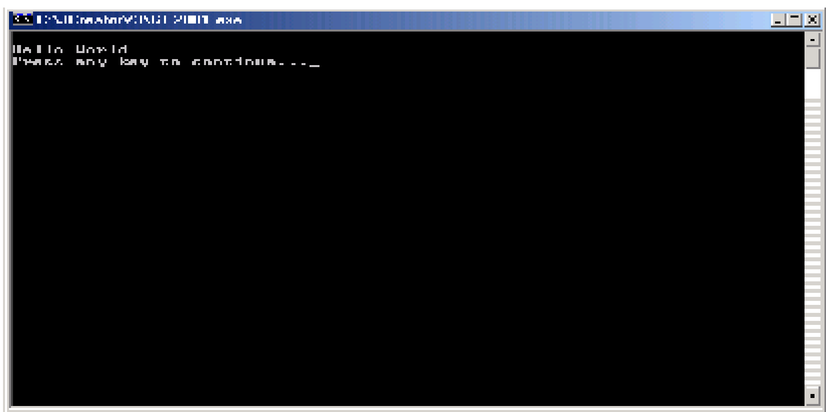
Công nghệ Java phát triển mạnh mẽ nhờ vào "đại gia Sun Microsystem" cung cấp nhiều công cụ, thư viện lập trình phong phú hỗ trợ cho việc phát triển nhiều loại hình ứng dụng khác nhau cụ thể như: J2SE (Java 2 Standard Edition) hỗ trợ phát triển những ứng dụng đơn, ứng dụng client-server; J2EE (Java 2 Enterprise Edition) hỗ trợ phát triển các ứng dụng thương mại, J2ME (Java 2 Micro Edition) hỗ trợ phát triển các ứng dụng trên các thiết bị di động, không dây, …

1.3.Các ứng dụng Java

1.3.1.Java và ứng dụng Console

Ứng dụng Console là ứng dụng nhập xuất ở chế độ văn bản tương tự như màn hình Console của hệ điều hành MS-DOS. Loại chương trình ứng dụng này thích hợp với những ai bước đầu làm quen với ngôn ngữ lập trình java.

Các ứng dụng kiểu Console thường được dùng để minh họa các ví dụ cơ bản liên quan đến cú pháp ngôn ngữ, các thuật toán, và các chương trình ứng dụng không cần thiết đến giao diện người dùng đồ họa.

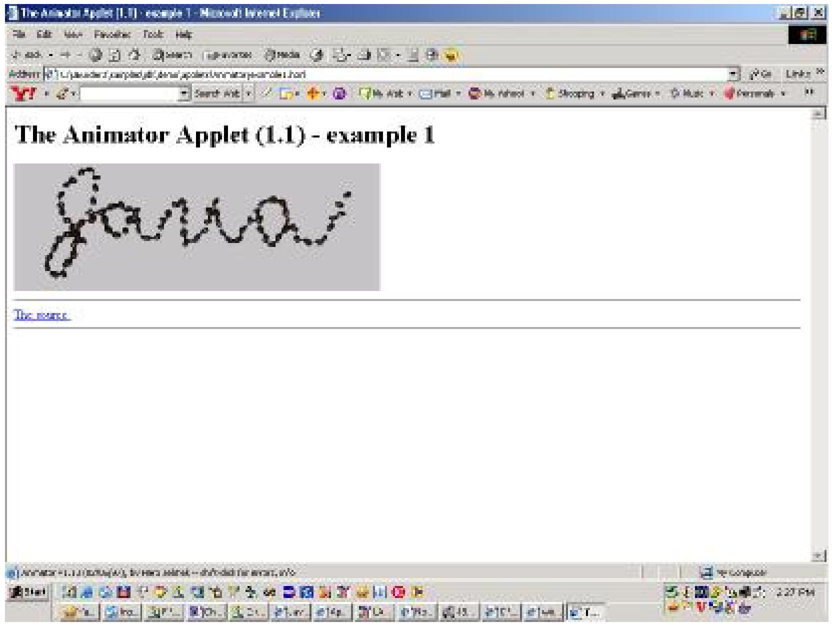


class HelloWorld
{ public static void main(String[] args)

```
	{
		System.out.println("\nHello World");
	}
}
```

1.3.2.Java và ứng dụng Applet

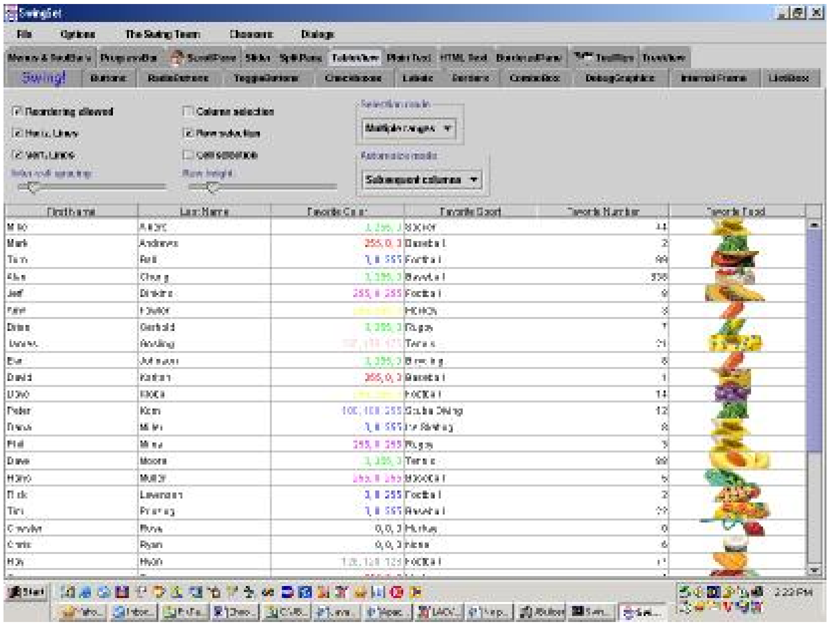
Java Applet là loại ứng dụng có thể nhúng và chạy trong trang web của một trình duyệt web. Từ khi internet mới ra đời, Java Applet cung cấp một khả năng lập trình mạnh mẽ cho các trang web. Nhưng gần đây khi các chương trình duyệt web đã phát triển với khả năng lập trình bằng VB Script, Java Script, HTML, DHTML, XML, … cùng với sự canh tranh khốc liệt của Microsoft và Sun đã làm cho Java Applet lu mờ. Và cho đến bây giờ gần như các lập trình viên đều không còn "mặn mà" với Java Applet nữa. (trình duyệt IE đi kèm trong phiên bản Windows 2000 đã không còn mặc nhiên hỗ trợ thực thi một ứng dụng Java Applet). Hình bên dưới minh họa một chương trình java applet thực thi trong một trang web.



1.3.3.Java và phát triển ứng dụng Desktop dùng AWT và JFC

Việc phát triển các chương trình ứng dụng có giao diện người dùng đồ họa trực quan giống như những chương trình được viết dùng ngôn ngữ lập trình VC++ hay Visual Basic đã được java giải quyết bằng thư viện AWT và JFC. JFC là thư viện rất phong phú và hỗ trợ mạnh mẽ hơn nhiều so với AWT. JFC giúp cho người lập trình có thể tạo ra một giao diện trực quan của bất kỳ ứng dụng nào. Liên quan đến việc phát triển các ứng dụng có giao diện người dùng đồ họa trực quan chúng ta sẽ tìm hiểu chi tiết trong chương 4.

Minh họa thiết kế giao diện người dùng sử dụng JFC



1.3.4.Java và phát triển ứng dụng Web

Java hỗ trợ mạnh mẽ đối với việc phát triển các ứng dụng Web thông qua công nghệ J2EE (Java 2 Enterprise Edition). Công nghệ J2EE hoàn toàn có thể tạo ra các ứng dụng Web một cách hiệu quả không thua kém công nghệ .NET mà Microsft đang quảng cáo.

Hiện nay có rất nhiều trang Web nổi tiếng ở Việt Nam cũng như khắp nơi trên thế giới được xây dựng và phát triển dựa trên nền công nghệ Java. Số ứng dụng Web được xây dựng dùng công nghệ Java chắc chắn không ai có thể biết được con số chính xác là bao nhiêu, nhưng chúng tôi đưa ra đây vài ví dụ để thấy rằng công nghệ Java của Sun là một “đối thủ đáng gờm” của Microsoft.

http://java.sun.com/

http://e-docs.bea.com/

http://www.macromedia.com/software/jrun/

http://tomcat.apache.org/index.html

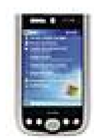
Chắc không ít người trong chúng ta biết đến trang web thông tin nhà đất nổi tiếng ở TPHCM đó là: http://www.nhadat.com/. Ứng dụng Web này cũng được xây dựng dựa trên nền công nghệ java.

Bạn có thể tìm hiểu chi tiết hơn về công nghệ J2EE tạo địa chỉ:

http://java.sun.com/j2ee/

1.3.5.Java và phát triển các ứng dụng nhúng

Java Sun đưa ra công nghệ J2ME (The Java 2 Platform, Micro Edition J2ME) hỗ trợ phát triển các chương trình, phần mềm nhúng. J2ME cung cấp một môi trường cho những chương trình ứng dụng có thể chạy được trên các thiết bị cá nhân như: điện thoại di động, máy tính bỏ túi PDA hay Palm, cũng như các thiết bị nhúng khác.

Bạn có thể tìm hiểu chi tiết hơn về công nghệ J2ME tại địa chỉ:

http://java.sun.com/j2me/

1.4.Dịch và thực thi một chương trình viết bằng Java

Việc xây dựng, dịch và thực thi một chương trình viết bằng ngôn ngữ lập trình java có thể tóm tắt qua các bước sau:

- **Viết mã nguồn**: dùng một chương trình soạn thảo nào đấy (NotePad hay Jcreator chẳng hạn) để viết mã nguồn và lưu lại với tên có đuôi ".java"

- **Biên dịch ra mã máy ảo**: dùng trình biên dịch javac để biên dịch mã nguồn ".java" thành mã của máy ảo (java bytecode) có đuôi ".class" và lưu lên đĩa
- **Thông dịch và thực thi**: ứng dụng được load vào bộ nhớ, thông dịch và thực thi dùng trình thông dịch Java thông qua lệnh "java".
    - **Đưa mã java bytecode vào bộ nhớ**: đây là bước "*loading*". Chương trình phải được đặt vào trong bộ nhớ trước khi thực thi. "Loader" sẽ lấy các files chứa mã java bytecode có đuôi ".class" và nạp chúng vào bộ nhớ.
    - **Kiểm tra mã java bytecode**: trước khi trình thông dịch chuyển mã bytecode thành mã máy tương ứng để thực thi thì các mã bytecode phải được kiểm tra tính hợp lệ.
    - **Thông dịch & thực thi**: cuối cùng dưới sự điều khiển của CPU và trình thông dịch tại mỗi thời điểm sẽ có một mã bytecode được chuyển sang mã máy và thực thi.

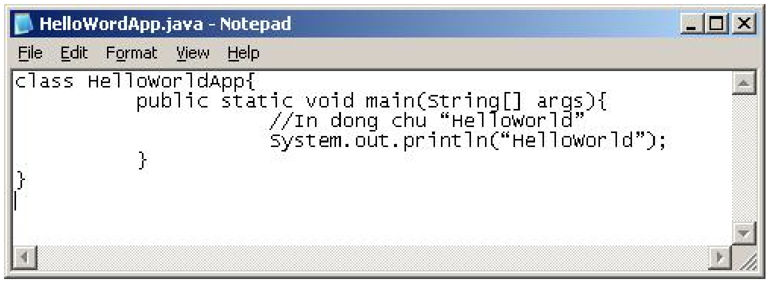
1.5.Chương trình Java đầu tiên

1.5.1.Tạo chương trình nguồn HelloWordApp

•Khởi động Notepad và gõ đoạn mã sau

```
/*Viết chương trình in dòng HelloWorld lên màn hình
Console*/
class HelloWorldApp{
	public static void main(String[] args){
		//In dong chu "HelloWorld"
		System.out.println("HelloWorld");
	}
}
```

Lưu lại với tên HelloWorldApp.java



1.5.2.Biên dịch tập tin nguồn HelloWordApp

Việc biên dịch tập tin mã nguồn chương trình HelloWorldApp có thể thực hiện qua các bước cụ thể như sau:

- Mở cửa sổ Command Prompt.
- Chuyển đến thư mục chứa tập tin nguồn vừa tạo ra.
- Thực hiện câu lệnh: **javac HelloWordApp.java**

Nếu gặp thông báo lỗi "Bad Command of filename" hoặc "The name specified is not recognized as an internal or external command, operable program or batch file" có nghĩa là Windows không tìm được trình biên dịch javac. Để sửa lỗi này chúng ta cần cập nhật lại đường dẫn PATH của hệ thống. Ngược lại nếu thành công bạn sẽ có thêm tập tin HelloWordApp.class

1.5.3.Chạy chương trình HelloWordApp

- Tại dấu nhắc gõ lệnh: **java HelloWordApp**
- Nếu chương trình đúng bạn sẽ thấy dòng chữ HelloWord trên màn hình Console.
- Nếu các bạn nhận được lỗi "Exception in thread "main java.lang.NoClassDefFoundError: HelloWorldApp" có nghĩa là Java không thể tìm được tập tin mã bytecode tên HelloWorldApp.class của các bạn. Một trong những nơi java cố tìm tập tin bytecode là thư mục hiện tại của

các bạn. Vì thế nếu tập tin byte code được đặt ở C:\java thì các bạn nên thay đổi đường dẫn tới đó.

1.5.4.Cấu trúc chương trình HelloWordApp

Phương thức main(): là điểm bắt đầu thực thi một ứng dụng. Mỗi ứng dụng Java phải chứa một phương thức main có dạng như sau: **public static void main(String[] args)**
Phương thức main chứa ba bổ từ đặc tả sau:

•**public** chỉ ra rằng phương thức main có thể được gọi bỡi bất kỳ đối tượng nào.

•**static** chỉ ra rằng phương thức main là một phương thức lớp.

•**void** chỉ ra rằng phương thức main sẽ không trả về bất kỳ một giá trị nào.

Ngôn ngữ Java hỗ trợ ba kiểu chú thích sau:
•/* *text* */
•// text
•/** documentation */. Công cụ javadoc trong bộ JDK sử dụng chú thích này để chuẩn bị cho việc tự động phát sinh tài liệu.

- Dấu mở và đóng ngo8ạc nhọn “{“ và “}”: là bắt đầu và kết thúc 1 khối lệnh.
- Dấu chấm phẩy “;” kết thúc 1 dòng lệnh.

1.5.5.Sử dụng phương thức/biến của lớp

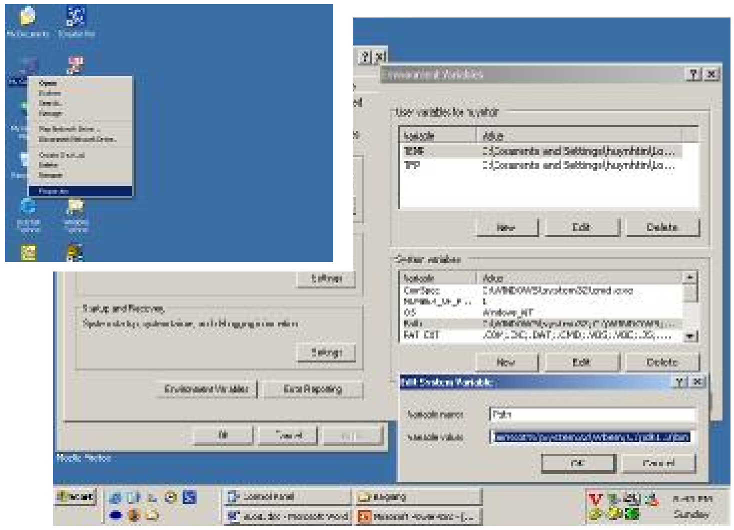
Cú pháp: Tên_lớp.Tên_biến
hoặc Tên_lớp.Tên_phương_thức(…)

1.6.Công cụ lập trình và chương trình dịch

1.6.1.J2SDK

- Download J2SE phiên bản mới nhất tương ứng với hệ điều hành đang sử dụng từ địa chỉ java.sun.com và cài

đặt lên máy tính (phiên bản được chúng tôi sử dụng khi viết giáo trình này là J2SE 1.4). Sau khi cài xong, chúng ta cần cập nhật đường dẫn PATH hệ thống chỉ đến thư mục chứa chương trình dịch của ngôn ngữ java.



1.6.2.Công cụ soạn thảo mã nguồn Java.

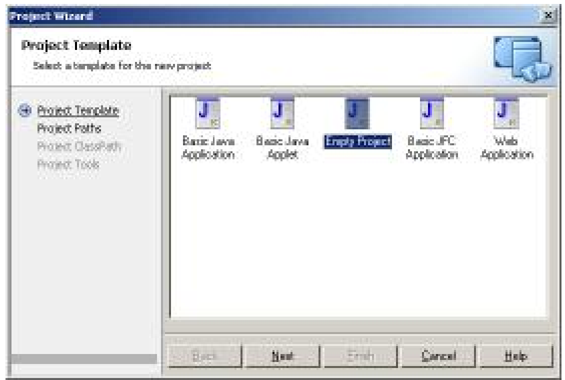
Để viết mã nguồn java chúng ta có thể sử dụng trình soạn thảo NotePad hoặc một số môi trường phát triển hỗ trợ ngôn ngữ java như: Jbuilder của hãng Borland, Visual Café của hãng Symantec, JDeveloper của hãng Oracle, Visual J++ của Microsoft, …

Trong khuôn khổ giáo trình này cũng như để hướng dẫn sinh viên thực hành chúng tôi dùng công cụ JCreator LE v3.50 của hãng XINOX Software. Các bạn có thể download JCreator LE v3.50 từ http://www.jcreator.com/download.htm.

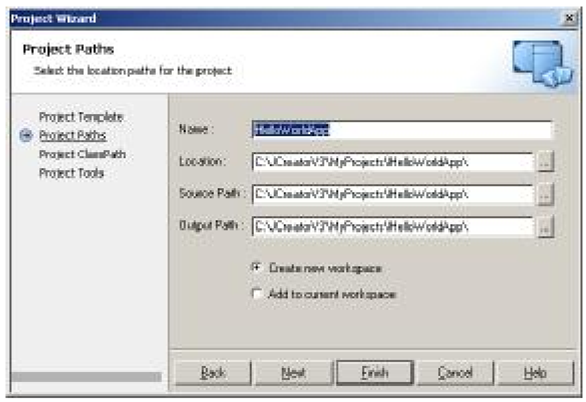
***Ví dụ**: Dùng JCreator tạo và thực thi chương trình có tên HelloWorldApp.*

<u>Bước 1</u>: Tạo 1 Empty Project

- *File ® New ® Project.*
- *Chọn Empty project rồi bấm nút chọn Next*

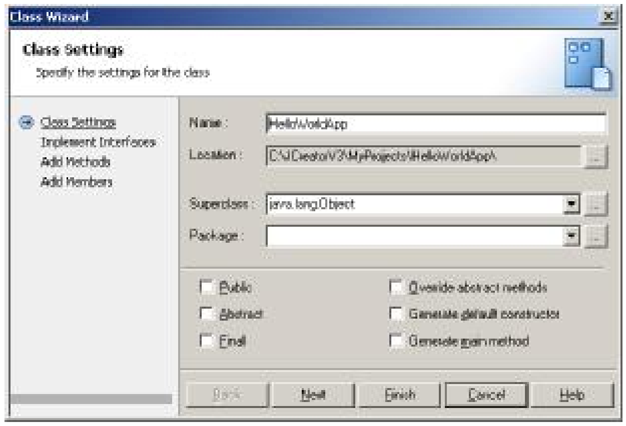


- *Sau đó nhập tên project và bấm chọn Finish.*

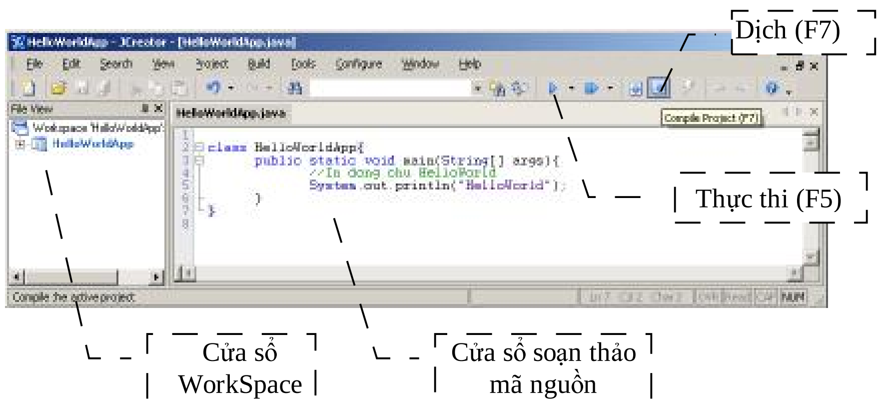


<u>Bước 2</u>: Tạo một Class mới tên HelloWorldApp và đưa vào Project hiện tại.

- *File ® New ® Class.*
- *Nhập vào tên Class và chọn Finish (hình bên dưới).*



Bước 3: Soạn thảo mã nguồn (hình bên dưới)



Chương 2:

HẰNG, BIẾN, KIỂU DỮ LIỆU, TOÁN TỬ, BIỂU THỨC VÀ CÁC CẤU TRÚC ĐIỀU KHIỂN TRONG JAVA

2.1.Biến

- Biến là vùng nhớ dùng để lưu trữ các giá trị của chương trình. Mỗi biến gắn liền với một kiểu dữ liệu và một định danh duy nhất gọi là tên biến.
- Tên biến thông thường là một chuỗi các ký tự (Unicode), ký số.
    - Tên biến phải bắt đầu bằng một chữ cái, một dấu gạch dưới hay dấu dollar.
    - Tên biến không được trùng với các từ khóa (xem phụ lục các từ khóa trong java).
    - Tên biến không có khoảng trắng ở giữa tên.
- Trong java, biến có thể được khai báo ở bất kỳ nơi đâu trong chương trình.

Cách khai báo

<kiểu_dữ_liệu> <tên_biến>;

<kiểu_dữ_liệu> <tên_biến> = <giá_trị>;

Gán giá trị cho biến

<tên_biến> = <giá_trị>;

Biến công cộng (toàn cục): là biến có thể truy xuất ở khắp nơi trong chương trình, thường được khai báo dùng từ khóa public, hoặc đặt chúng trong một class.

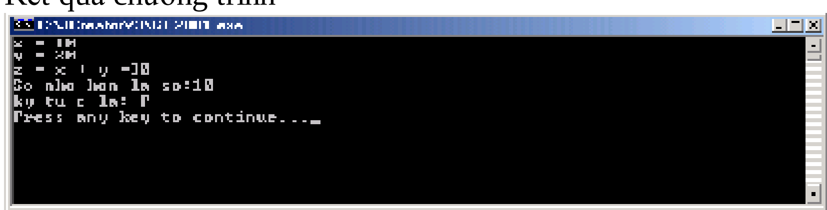
Biến cục bộ: là biến chỉ có thể truy xuất trong khối lệnh nó khai báo.

<u>Lưu ý</u>: Trong ngôn ngữ lập trình java có phân biệt chữ in hoa và in thường. Vì vậy chúng ta cần lưu ý khi đặt tên cho các đối tương dữ liệu cũng như các xử lý trong chương trình.

Ví dụ:

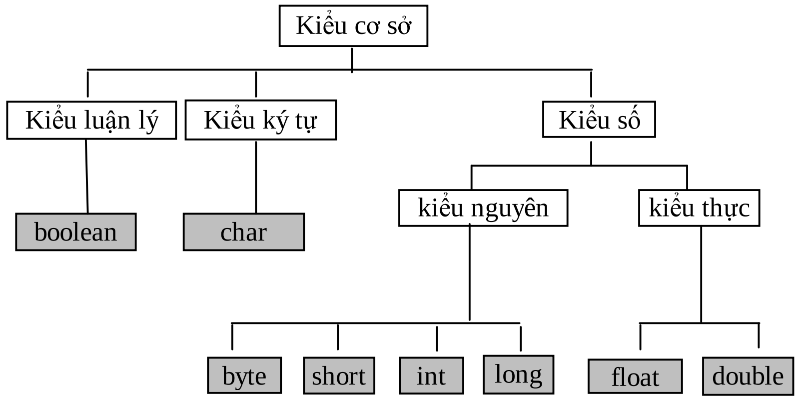
```
import java.lang.*;
import java.io.*;
class VariableDemo
{
        static int x, y;
        public static void main(String[] args)
        {
                x = 10;
                y = 20;
                int z = x+y;
                System.out.println("x = " + x);
                System.out.println("y = " + y);
                System.out.println("z = x + y =" + z);
                System.out.println("So nho hon la so:" +
                Math.min(x, y));
                char c = 80;
                System.out.println("ky tu c la: " + c);
        }
}
```

Kết quả chương trình



2.2.Các kiểu dữ liệu cơ sở

Ngôn ngữ lập trình java có 8 kiểu dữ liệu cơ sở: byte, short, int, long, float, double, boolean và char.



Kiểu	Kích thước (bytes)	Giá trị min	Giá trị max	Giá trị mặc định
byte	1	-256	255	0
short	2	-32768	32767	0
int	4	-2^{31}	$2^{31} - 1$	0
long	8	-2^{63}	$2^{63} - 1$	0L
float	4			0.0f
double	8			0.0d

2.2.1.Kiểu số nguyên

- Java cung cấp 4 kiểu số nguyên khác nhau là: byte, short, int, long. Kích thước, giá trị nhỏ nhất, lớn nhất, cũng như giá trị mặc định của các kiểu dữ liệu số nguyên được mô tả chi tiết trong bảng trên.
- Kiểu mặc định của các số nguyên là kiểu int.
- Các số nguyên kiểu byte và short rất ít khi được dùng.
- Trong java không có kiểu số nguyên không dấu như trong ngôn ngữ C/C++.

Khai báo và khởi tạo giá trị cho các biến kiểu nguyên:

```
int x = 0;
long y = 100;
```

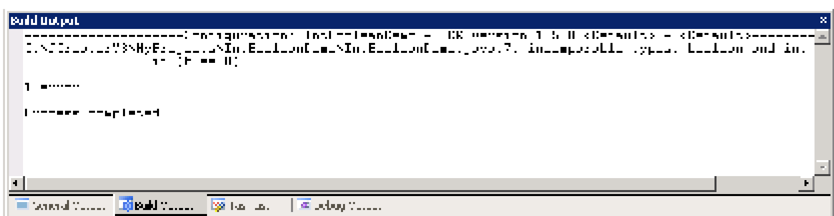
Một số lưu ý đối với các phép toán trên số nguyên:

- Nếu hai toán hạng kiểu long thì kết quả là kiểu long. Một trong hai toán hạng không phải kiểu long sẽ được chuyển thành kiểu long trước khi thực hiện phép toán.
- Nếu hai toán hạng đầu không phải kiểu long thì phép tính sẽ thực hiện với kiểu int.
- Các toán hạng kiểu byte hay short sẽ được chuyển sang kiểu int trước khi thực hiện phép toán.
- Trong java không thể chuyển biến kiểu int và kiểu boolean như trong ngôn ngữ C/C++.

Ví dụ: có đoạn chương trình như sau

```
boolean b = false;
if (b == 0)
{
    System.out.println("Xin chao");
}
```

Lúc biên dịch đoạn chương trình trên trình dịch sẽ báo lỗi: không được phép so sánh biến kiểu boolean với một giá trị kiểu int.



2.2.2.Kiểu dấu chấm động

Đối với kiểu dấu chấm động hay kiểu thực, java hỗ trợ hai kiểu dữ liệu là float và double.
Kiểu float có kích thước 4 byte và giá trị mặc định là 0.0f
Kiểu double có kích thước 8 byte và giá trị mặc định là 0.0d

Số kiểu dấu chấm động không có giá trị nhỏ nhất cũng không có giá trị lớn nhất. Chúng có thể nhận các giá trị:

- Số âm
- Số dương
- Vô cực âm
- Vô cực dương

Khai báo và khởi tạo giá trị cho các biến kiểu dấu chấm động:

```
float x = 100.0/7;
double y = 1.56E6;
```

Một số lưu ý đối với các phép toán trên số dấu chấm động:

- Nếu mỗi toán hạng đều có kiểu dấn chấm động thì phép toán chuyển thành phép toán dấu chấm động.
- Nếu có một toán hạng là double thì các toán hạng còn lại sẽ được chuyển thành kiểu double trước khi thực hiện phép toán.
- Biến kiểu float và double có thể ép chuyển sang kiểu dữ liệu khác trừ kiểu boolean.

2.2.3.Kiểu ký tự (char)

Kiểu ký tự trong ngôn ngữ lập trình java có kích thước là 2 bytes và chỉ dùng để biểu diễn các ký tự trong bộ mã Unicode. Như vậy kiểu char trong java có thể biểu diễn tất cả 2^{16} = 65536 ký tự khác nhau.
Giá trị mặc định cho một biến kiểu char là null.

2.2.4.Kiểu luận lý (boolean)

- Kiểu boolean chỉ nhận 1 trong 2 giá trị: **true** hoặc **false**.
- Trong java kiểu boolean không thể chuyển thành kiểu nguyên và ngược lại.
- Giá trị mặc định của kiểu boolean là false.

2.3.Hằng:

- Hằng là một giá trị bất biến trong chương trình
- Tên hằng được đặt theo qui ước giống như tên biến.
- Hằng số nguyên: trường hợp giá trị hằng ở dạng long ta thêm vào cuối chuỗi số chữ "**l**" hay "**L**". (ví dụ: **1L**)
- Hằng số thực: truờng hợp giá trị hằng có kiểu float ta thêm tiếp vĩ ngữ "**f**" hay "**F**", còn kiểu số double thì ta thêm tiếp vĩ ngữ "**d**" hay "**D**".
- Hằng Boolean: java có 2 hằng boolean là **true**, **false**.
- Hằng ký tự: là một ký tự đơn nằm giữa nằm giữa 2 dấu ngoặc đơn.
  - Ví dụ: 'a': hằng ký tự a
  - Một số hằng ký tự đặc biệt

Ký tự	Ý nghĩa
\b	Xóa lùi (BackSpace)
\t	Tab
\n	Xuống hàng
\r	Dấu enter
\"	Nháy kép
\'	Nháy đơn
\\	Số ngược
\f	Đẩy trang
\uxxxx	Ký tự unicode

- Hằng chuỗi: là tập hợp các ký tự được đặt giữa hai dấu nháy kép “”. Một hằng chuỗi không có ký tự nào là một hằng chuỗi rỗng.
    - Ví dụ: “Hello Wolrd”
    - Lưu ý: Hằng chuỗi không phải là một kiểu dữ liệu cơ sở nhưng vẫn được khai báo và sử dụng trong các chương trình.

2.4.Lệnh, khối lệnh trong java

Giống như trong ngôn ngữ C, các câu lệnh trong java kết thúc bằng một dấu chấm phẩy (;).

Một khối lệnh là đoạn chương trình gồm hai lệnh trở lên và được bắt đầu bằng dấu mở ngoặc nhọn ({) và kết thúc bằng dấu đóng ngoặc nhọc (}).

Bên trong một khối lệnh có thể chứa một hay nhiều lệnh hoặc chứa các khối lệnh khác.

```
{ // khối 1
        { // khối 2
                lệnh 2.1
                lệnh 2.2
                …
        } // kết thúc khối lệnh 2
        lệnh 1.1
        lệnh 1.2
        …
} // kết thúc khối lệnh 1

{ // bắt đầu khối lệnh 3
        // Các lệnh thuộc khối lệnh 3
        // …
} // kết thúc thối lệnh 3
```

2.5.Toán tử và biểu thức

2.5.1.Toán tử số học

Toán tử	Ý nghĩa
+	Cộng
-	Trừ
*	Nhân
/	Chia nguyên
%	Chia dư
++	Tăng 1
--	Giảm 1

2.5.2.Toán tử trên bit

Toán tử	Ý nghĩa
&	AND
|	OR
^	XOR
<<	Dịch trái
>>	Dịch phải
>>>	Dịch phải và điền 0 vào bit trống
~	Bù bit

2.5.3.Toán tử quan hệ & logic

Toán tử	Ý nghĩa
==	So sánh bằng
!=	So sánh khác
>	So sánh lớn hơn
<	So sánh nhỏ hơn
>=	So sánh lớn hơn hay bằng
<=	So sánh nhỏ hơn hay bằng

||	OR (biểu thức logic)
&&	AND (biểu thức logic)
!	NOT (biểu thức logic)

2.5.4.Toán tử ép kiểu

- Ép kiểu rộng (widening conversion): từ kiểu nhỏ sang kiểu lớn (không mất mát thông tin)
- Ép kiểu hẹp (narrow conversion): từ kiểu lớn sang kiểu nhỏ (có khả năng mất mát thông tin)

<tên biến> = (kiểu_dữ_liệu) <tên_biến>;

Ví dụ:

float fNum = 2.2;
int iCount = (int) fNum; // (iCount = 2)

2.5.5.Toán tử điều kiện

Cú pháp: <điều kiện> ? <biểu thức 1> : < biểu thức 2>
Nếu điều kiện đúng thì có giá trị, hay thực hiện <biểu thức 1>, còn ngược lại là <biểu thức 2>.
<điều kiện>: là một biểu thức logic
<biểu thức 1>, <biểu thức 2>: có thể là hai giá trị, hai biểu thức hoặc hai hành động.

Ví dụ:

int x = 10;
int y = 20;
int Z = (x<y) ? 30 : 40;
// Kết quả z = 30 do biểu thức (x < y) là đúng.

2.5.6.Thứ tự ưu tiên

Thứ tự ưu tiên tính từ trái qua phải và từ trên xuống dưới

Cao nhất			

<table>
<tr><td>()</td><td>[]</td><td>.</td><td></td></tr>
<tr><td>++</td><td>--</td><td>~</td><td>!</td></tr>
<tr><td>*</td><td>/</td><td>%</td><td></td></tr>
<tr><td>+</td><td>-</td><td></td><td></td></tr>
<tr><td>>></td><td>>>> (dịch phải và điền 0 vào bit trống)</td><td><<</td><td></td></tr>
<tr><td>></td><td>>=</td><td><</td><td><=</td></tr>
<tr><td>==</td><td>!=</td><td></td><td></td></tr>
<tr><td>&</td><td></td><td></td><td></td></tr>
<tr><td>^</td><td></td><td></td><td></td></tr>
<tr><td>|</td><td></td><td></td><td></td></tr>
<tr><td>&&</td><td></td><td></td><td></td></tr>
<tr><td>||</td><td></td><td></td><td></td></tr>
<tr><td>?:</td><td></td><td></td><td></td></tr>
<tr><td>=</td><td><toántử>=</td><td></td><td></td></tr>
<tr><td>Thấp nhất</td><td></td><td></td><td></td></tr>
</table>

2.6.Cấu trúc điều khiển

2.6.1.Cấu trúc điều kiện if … else

Dạng 1:

```
if (<điều_kiện>)
{
        <khối_lệnh>;
}
```

Dạng 2:

```
if (<điều_kiện>)
{
        <khối _lệnh1>;
}
else
{
        <khối _lệnh2>;
```

```
}
```

2.6.2.Cấu trúc switch … case

```
switch (<biến>)
{
        case <giátrị_1>:
                <khối_lệnh_1>;
                break;
                ….
        case <giátrị_n>:
                <khối_lệnh_n>;
                break;
        default:
                <khối lệnh default>;
}
```

2.6.3.Cấu trúc lặp

Dạng 1: while(…)

```
while (điều_kiện_lặp)
{
        khối _lệnh;
}
```

Dạng 2: do { … } while;

```
do
{
        khối_lệnh;
} while (điều_kiện);
```

Dạng 3: for (…)

```
for (khởi_tạo_biến_đếm;đk_lặp;tăng_biến)
{
        <khối _lệnh>;
```

}

2.6.4.Cấu trúc lệnh nhảy (jump)

Lệnh break: trong cấu trúc switch chúng ta dùng câu lệnh break để thoát thỏi cấu trúc switch trong cùng chứa nó. Tương tự như vậy, trong cấu trúc lặp, câu lệnh break dùng để thóat khỏi cấu trúc lặp trong cùng chứa nó.

Lệnh continue: dùng để tiếp tục vòng lặp trong cùng chứa nó (ngược với break).

Nhãn (label):

Không giống như C/C++, Java không hỗ trợ lệnh goto để nhảy đến 1 vị trí nào đó của chương trình. **Java dùng kết hợp nhãn (label) với từ khóa break và continue** để thay thế cho lệnh **goto**.

Ví dụ:

```
label:
for (…)
{         for (…)
          {         if (<biểu thức điều kiện>)
                              break label;
                    else
                              continue label;
          }
}
```

Lệnh "*label:*" xác định vị trí của nhãn và xem như tên của vòng lặp ngoài. Nếu <*biểu thức điều kiện*> đúng thì lệnh *break label* sẽ thực hiện việc nhảy ra khỏi vòng lặp có nhãn là "*label*", ngược lại sẽ tiếp tục vòng lặp có nhãn "*label*" (khác với *break* và *continue* thông thường chỉ thoát khỏi hay tiếp tục vòng lặp trong cùng chứa nó.).

2.7.Lớp bao kiểu dữ liệu cơ sở (Wrapper Class)

Data type	Wrapper Class	Ghi chú

<table>
<tr><td></td><td>(java.lang.*)</td><td></td></tr>
<tr><td>boolean</td><td>Boolean</td><td rowspan="8">- Gói (package): chứa nhóm nhiều class.
- Ngoài các Wrapper Class, gói java.lang còn cung cấp các lớp nền tảng cho việc thiết kế ngôn ngữ java như: String, Math, …</td></tr>
<tr><td>byte</td><td>Byte</td></tr>
<tr><td>short</td><td>Short</td></tr>
<tr><td>char</td><td>Character</td></tr>
<tr><td>int</td><td>Integer</td></tr>
<tr><td>long</td><td>Long</td></tr>
<tr><td>Float</td><td>Float</td></tr>
<tr><td>double</td><td>Double</td></tr>
</table>

2.8.Kiểu dữ liệu mảng

Như chúng ta đã biết Java có 2 kiểu dữ liệu

- Kiểu dữ liệu cơ sở (Primitive data type)
- Kiểu dữ liệu tham chiếu hay dẫn xuất (reference data type): thường có 3 kiểu:
  - Kiểu mảng
  - Kiểu lớp
  - Kiểu giao tiếp(interface).

Ở đây chúng ta sẽ tìm hiểu một số vấn đề cơ bản liên quan đền kiểu mảng. Kiểu lớp(class) và giao tiếp(interface) chúng ta sẽ tìm hiểu chi tiết trong chương 3 và các chương sau.

2.8.1.Khái niệm mảng

Mảng là tập hợp nhiều phần tử có cùng tên, cùng kiểu dữ liệu và mỗi phần tử trong mảng được truy xuất thông qua chỉ số của nó trong mảng.

2.8.2.Khai báo mảng

```
        <kiểu dữ liệu>          <tên mảng>[];
hoặc    <kiểu dữ liệu>[]        <tên mảng>;
```

***Ví dụ*:**

```
        int     arrInt[];
hoặc    int[]   arrInt;
```

int[]   arrInt1, arrInt2, arrInt3;

2.8.3.Cấp phát bộ nhớ cho mảng

- Không giống như trong C, C++ kích thước của mảng được xác định khi khai báo. Chẳng hạn như:

int arrInt[100]; // Khai báo náy trong Java sẽ bị báo lỗi.

- Để cấp phát bộ nhớ cho mảng trong Java ta cần dùng từ khóa **new**. (Tất cả trong Java đều thông qua các đối tượng). Chẳng hạn để cấp phát vùng nhớ cho mảng trong Java ta làm như sau:

*int arrInt = **new** int[100];*

2.8.4.Khởi tạo mảng

Chúng ta có thể khởi tạo giá trị ban đầu cho các phần tử của mảng khi nó được khai báo.

Ví dụ:

int arrInt[] = {1, 2, 3};
char arrChar[] = {'a', 'b', 'c'};
String arrStrng[] = {"ABC", "EFG", 'GHI'};

2.8.5.Truy cập mảng

Chỉ số mảng trong Java bắt đầu tư 0. Vì vậy phần tử đầu tiên có chỉ số là 0, và phần tử thứ n có chỉ số là n-1. Các phần tử của mảng được truy xuất thông qua chỉ số của nó đặt giữa cặp dấu ngoặc vuông ([]).

Ví dụ:

int arrInt[] = {1, 2, 3};
int x = arrInt[0]; // x sẽ có giá trị là 1.
int y = arrInt[1]; // y sẽ có giá trị là 2.
int z = arrInt[2]; // z sẽ có giá trị là 3.

Lưu ý: Trong nhưng ngôn ngữ lập trình khác (C chẳng hạn), một chuỗi được xem như một mảng các ký tự. Trong java thì

khác, java cung cấp một lớp String để làm việc với đối tượng dữ liệu chuỗi cùng khác thao tác trên đối tượng dữ liệu này.

2.9.Một số ví dụ minh họa:

Ví dụ 1: Nhập ký tự từ bàn phím

```
import java.io.*;
/* gói này cung cấp thự viện xuất nhập hệ thống thông qua
những luồng dữ //liệu và hệ thống file.*/
class InputChar
{
	public static void main(String args[])
	{
		char ch = ‘’;
		try
		{
			ch = (char) System.in.read();
		}
		catch(Exception e)
		{
			System.out.println(“Nhập lỗi!”);
		}

		System.out.println(“Ky tu vua nhap:” + ch);
	}
}
```

Ví dụ 2: Nhập dữ liệu số

```
import java.io.*;
class inputNum
{	public static void main(String[] args)
	{	int n=0;
		try
		{	BufferedReader in =
				new BufferedReader(
```

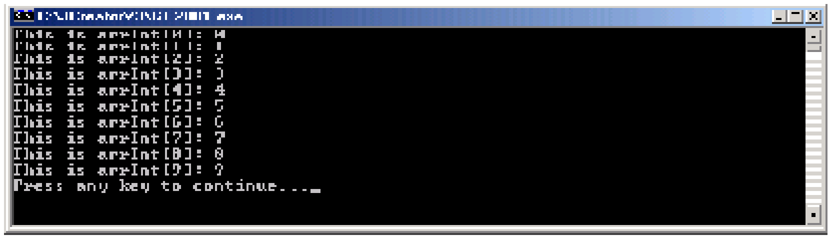
```
                                        new InputStreamReader(
                                        System.in));
                        String s;
                        s = in.readLine();
                        n = Integer.parseInt(s);
                }
                catch(Exception e)
                {       System.out.println("Nhập dữ liệu bị
                                                lỗi !");
                }

                System.out.println("Bạn vừa nhập số:" + n);
        }
}
```

Ví dụ 3: Nhập và xuất giá trị các phần tử của một mảng các số nguyên.

```
class ArrayDemo
{
        public static void main(String args[])
        {
                int arrInt[] = new int[10];
                int i;
                for(i = 0; i < 10; i = i+1)
                        arrInt[i] = i;

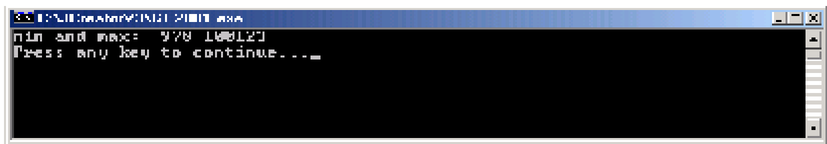
                for(i = 0; i < 10; i = i+1)
                        System.out.println("This is arrInt[" + i +
                        "]: " + arrInt[i]);
        }
}
```



Ví dụ 4: Tìm phần tử có giá trị nhỏ nhất (Min) và lớn nhất (Max) trong một mảng.

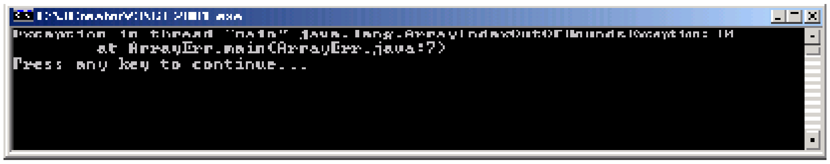
```
class MinMax
{	public static void main(String args[])
	{	int nums[] = new int[10];
		int min, max;
		nums[0] = 99;
		nums[1] = -10;
		nums[2] = 100123;
		nums[3] = 18;
		nums[4] = -978;
		nums[5] = 5623;
		nums[6] = 463;
		nums[7] = -9;
		nums[8] = 287;
		nums[9] = 49;
		min = max = nums[0];
		for(int i=1; i < 10; i++)
		{
			if(nums[i] < min) min = nums[i];
			if(nums[i] > max) max = nums[i];
		}
		System.out.println("min and max: " + min + " "
		+ max);
	}
}
class MinMax2
```

```
{
        public static void main(String args[])
        {
                int nums[] = { 99, -10, 100123, 18, -978,
                                5623, 463, -9, 287, 49 };
                int min, max;
                min = max = nums[0];
                for(int i=1; i < 10; i++)
                {
                        if(nums[i] < min) min = nums[i];
                        if(nums[i] > max) max = nums[i];
                }
                System.out.println("Min and max: " + min + " "
                + max);
        }
}
```



Ví dụ 5: chương trình minh họa một lỗi tham chiếu đến phần tử bên ngoài (vượt quá) kích thước mảng.

```
class ArrayErr
{       public static void main(String args[])
        {       int sample[] = new int[10];
                int i;
                for(i = 0; i < 100; i = i+1)
                        sample[i] = i;
        }
}
```



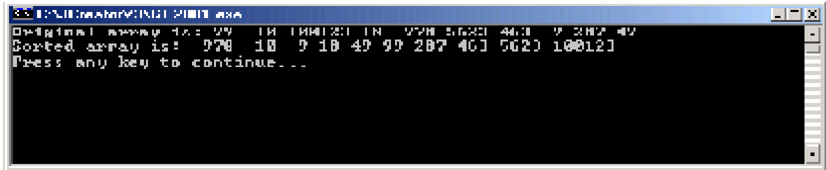
Ví dụ 6: Sắp xếp mảng dùng phương pháp sắp xếp nổi bọt (Bubble Sort)

```
class BubbleSort
{       public static void main(String args[])
        {       int nums[] = { 99, -10, 100123, 18, -978,
                                5623, 463, -9, 287, 49 };
                int a, b, t;
                int size;
                size = 10; // number of elements to sort
                // display original array
                System.out.print("Original array is:");
                for(int i=0; i < size; i++)
                        System.out.print(" " + nums[i]);

                System.out.println();
                // This is the Bubble sort.
                for(a=1; a < size; a++)
                        for(b=size-1; b >= a; b--)
                        {       if(nums[b-1] > nums[b])
                                {       // if out of order
                                        // exchange elements
                                        t = nums[b-1];
                                        nums[b-1] = nums[b];
                                        nums[b] = t;
                                }
                        }

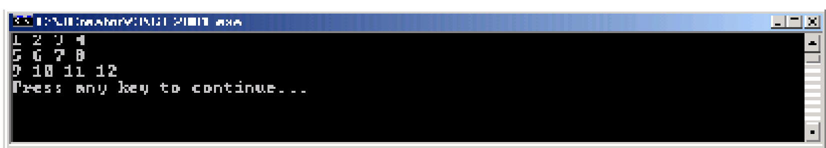
                // display sorted array
```

```
                System.out.print("Sorted array is:");
                for(int i=0; i < size; i++)
                        System.out.print(" " + nums[i]);
                System.out.println();
        }
}
```



Ví dụ 7: Nhập và xuất giá trị của các phần tử trong một mảng hai chiều.

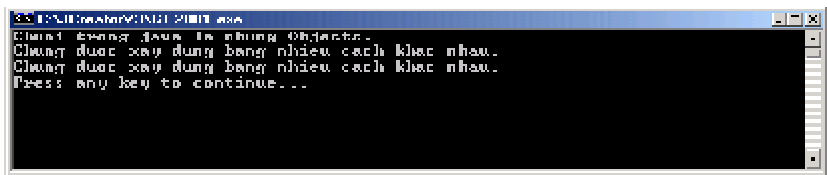
```
class TwoD_Arr
{       public static void main(String args[])
        {       int t, i;
                int table[][] = new int[3][4];
                for(t=0; t < 3; ++t)
                {       for(i=0; i < 4; ++i)
                        {       table[t][i] = (t*4)+i+1;
                                System.out.print(table[t][i]   +   "
                                ");
                        }
                        System.out.println();
                }
        }
}
```



Ví dụ 8: Tạo đối tượng chuỗi

```
class StringDemo
{
	public static void main(String args[])
	{
		// Tao chuoi bang nhieu cach khac nhau
		String str1 = new String("Chuoi trong java la nhung Objects.");
		String str2 = "Chung duoc xay dung bang nhieu cach khac nhau.";
		String str3 = new String(str2);

		System.out.println(str1);
		System.out.println(str2);
		System.out.println(str3);
	}
}
```



Ví dụ 9: Minh họa một số thao tác cơ bản trên chuỗi

```
// Chuong trinh minh hoa cac thao tac tren chuoi ky tu
class StrOps
{
	public static void main(String args[])
	{
		String str1 = "Java la chon lua so mot cho lap trinh ung dung Web.";
		String str2 = new String(str1);
		String str3 = "Java ho tro doi tuong String de xu ly chuoi";
```

```
int result, idx;
char ch;

System.out.println("str1:" + str1);
System.out.println("str2:" + str2);
System.out.println("str3:" + str3);

System.out.println("Chieu dai cua chuoi str1 la:
" + str1.length());

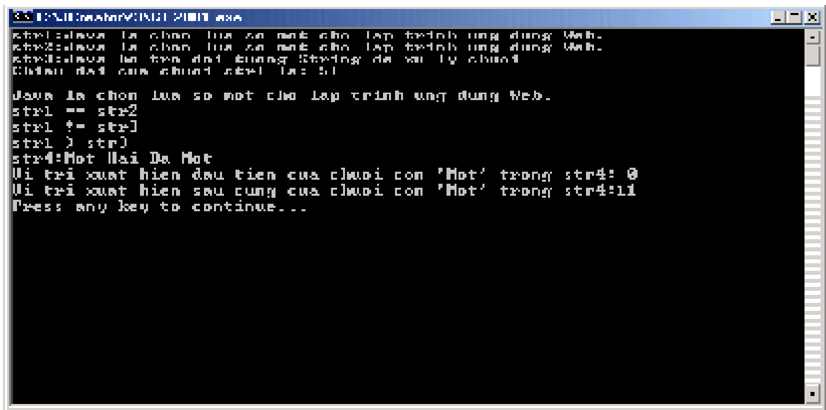
// Hien thi chuoi str1, moi lan mot ky tu.
System.out.println();
for(int i=0; i < str1.length(); i++)
        System.out.print(str1.charAt(i));

System.out.println();
if(str1.equals(str2))
        System.out.println("str1 == str2");
else
        System.out.println("str1 != str2");

if(str1.equals(str3))
        System.out.println("str1 == str3");
else
        System.out.println("str1 != str3");

result = str1.compareTo(str3);
if(result == 0)
        System.out.println("str1 = str3 ");
else
        if(result < 0)
                System.out.println("str1 < str3");
        else
                System.out.println("str1 > str3");
```

```
            // Tao chuoi moi cho str4
            String str4 = "Mot Hai Ba Mot";
            idx = str4.indexOf("Mot");
            System.out.println("str4:" + str4);
            System.out.println("Vi tri xuat hien dau tien cua
            chuoi con 'Mot' trong str4: " + idx);
            idx = str4.lastIndexOf("Mot");
            System.out.println("Vi tri xuat hien sau cung cua
            chuoi con 'Mot' trong str4:" + idx);
      }
}
```



Ví dụ 10: chương trình nhập vào một chuỗi và in ra chuỗi nghịch đảo của chuỗi nhập.

```
import java.lang.String;
import java.io.*;
public class InverstString
{     public static void main(String arg[])
      {     System.out.println("\n *** CHUONG TRINH IN
            CHUOI NGUOC *** ");
            try
```

```
                {       System.out.println("\n *** Nhap
                        chuoi:");
                        BufferedReader in = new
                        BufferedReader(new
                        InputStreamReader(System.in));
                        // Class BufferedReader cho phép đọc
                        text từ luồng nhập ký tự, tạo bộ đệm cho
                        những ký tự để hỗ trợ cho việc đọc những
                        ký tự, những mảng hay những dòng.

                        // Doc 1 dong tu BufferReadered ket thuc
                        bang dau ket thuc dong.
                        String str = in.readLine();
                        System.out.println("\n Chuoi vua nhap
                        la:" + str);

                        // Xuat chuoi nghich dao
                        System.out.println("\n Chuoi nghich dao
                        la:");
                        for (int i=str.length()-1; i>=0; i--)
                        {       System.out.print(str.charAt(i));
                        }
                }
                catch (IOException e)
                {       System.out.println(e.toString());
                }
        }
}
```

Ví dụ 11: Lấy chuỗi con của một chuỗi

```
class SubStr
{
        public static void main(String args[])
        {
```

```
                String orgstr = "Mot Hai Ba Bon";
                // Lay chuoi con dung ham
                // public String substring(int beginIndex, int
                // endIndex)
                String substr = orgstr.substring(4, 7);
                System.out.println("Chuoi goc: " + orgstr);
                System.out.println("Chuoi con: " + substr);
        }
}
```



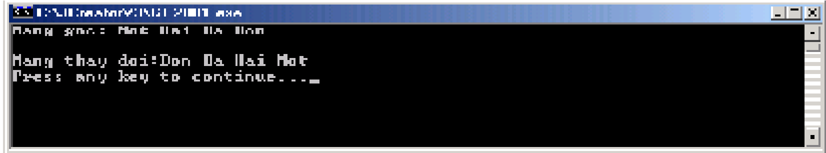
Ví dụ 12: Mảng các chuỗi

```
class StringArray
{
        public static void main(String args[])
        {
                String str[] = {"Mot", "Hai", "Ba", "Bon" };
                System.out.print("Mang goc: ");
                for(int i=0; i < str.length; i++)
                        System.out.print(str[i] + " ");

                System.out.println("\n");
                // Thay doi chuoi
                str[0] = "Bon";
                str[1] = "Ba";
                str[2] = "Hai";
                str[3] = "Mot";

                System.out.print("Mang thay doi:");
                for(int i=0; i < str.length; i++)
                        System.out.print(str[i] + " ");
```

```
            System.out.print("\n");
        }
}
```



Chương 3: HƯỚNG ĐỐI TƯỢNG TRONG JAVA

3.1.Mở đầu

Thông qua chuyên đề lập trình hướng đối tượng (OOP) chúng ta đã biết OOP là một trong những tiếp cận mạnh mẽ, và

rất hiệu quả để xây dựng nên những chương trình ứng dụng trên máy tính. Từ khi ra đời cho đến nay lập trình OOP đã chứng tỏ được sức mạnh, vai trò của nó trong các đề án tin học. Chương này sẽ giúp bạn đọc tìm hiểu về các kiểu dữ liệu dẫn xuất đó là lớp (class) và giao tiếp (interface), cũng như các vấn đề cơ bản về lập trình hướng đối tượng trong java thông qua việc tạo lập các lớp, các đối tượng và các tính chất của chúng.

3.2.Lớp (Class)

3.2.1.Khái niệm

Chúng ta có thể xem lớp như một khuôn mẫu (template) của đối tượng (Object). Trong đó bao gồm dữ liệu của đối tượng (fields hay properties) và các phương thức(methods) tác động lên thành phần dữ liệu đó gọi là các phương thức của lớp.

Các đối tượng được xây dựng bởi các lớp nên được gọi là các thể hiện của lớp (class instance).

3.2.2.Khai báo/định nghĩa lớp

```
class <ClassName>
{
        <kiểu dữ liệu> <field_1>;
        <kiểu dữ liệu> <field_2>;
        constructor
        method_1
        method_2
}
```

class: là từ khóa của java
ClassName: là tên chúng ta đặt cho lớp
field_1, field_2: các thuộc tính, các biến, hay các thành phần dữ liệu của lớp.
constructor: là sự xây dựng, khởi tạo đối tượng lớp.
method_1, method_2: là các phương thức/hàm thể hiện các thao tác xử lý, tác động lên các thành phần dữ liệu của lớp.

3.2.3.Tạo đối tượng của lớp

ClassName objectName = **new** *ClassName()*;

3.2.4.Thuộc tính của lớp

Vùng dữ liệu (fields) hay thuộc tính (properties) của lớp được khai báo bên trong lớp như sau:

```
class <ClassName>
{
        // khai báo những thuộc tính của lớp
        <tiền tố> <kiểu dữ liệu> field1;
        // …
}
```

Để xác định quyền truy xuất của các đối tượng khác đối với vùng dữ liệu của lớp người ta thường dùng 3 tiền tố sau:

- **public**: có thể truy xuất từ tất cả các đối tượng khác
- **private**: một lớp không thể truy xuất vùng private của 1 lớp khác.
- **protected**: vùng protected của 1 lớp chỉ cho phép bản thân lớp đó và những lớp dẫn xuất từ lớp đó truy cập đến.

Ví dụ:

```
public class xemay
{       public          String  nhasx;
        public          String  model;
        private         float   chiphisx;
        protected       int     thoigiansx;

        // so luong so cua xe may: 3, 4 so
        protected       int     so;

        // sobanhxe là biến tĩnh có giá trị là 2 trong tất cả
        // các thể hiện tạo ra từ lớp xemay
```

public static int sobanhxe = 2;
}

Thuộc tính “*nhasx”, “model”*có thể được truy cập đến từ tất cả các đối tượng khác.

Thuộc tính “*chiphisx”* chỉ có thể truy cập được từ các đối tượng có kiểu “*xemay”*

Thuộc tính “*thoigiansx”*, so có thể truy cập được từ các đối tượng có kiểu “*xemay”* và các đối tượng của các lớp con dẫn xuất từ lớp *“xemay”*

Lưu ý: Thông thường để an toàn cho vùng dữ liệu của các đối tượng người ta tránh dùng tiền tố public, mà thường chọn tiền tố private để ngăn cản quyền truy cập đến vùng dữ liệu của một lớp từ các phương thức bên ngoài lớp đó.

3.2.5.Hàm - Phương thức lớp (Method)

Hàm hay phương thức (method) trong Java là khối lệnh thực hiện các chức năng, các hành vi xử lý của lớp lên vùng dữ liệu.

Khai báo phương thức:

<Tiền tố> <kiểu trả về> <Tên phương thức> (<danh sách đối số>)
{
<khối lệnh>;
}

Để xác định quyền truy xuất của các đối tượng khác đối với các phương thức của lớp người ta thường dùng các tiền tố sau:

- **public**: phương thức có thể truy cập được từ bên ngoài lớp khai báo.
- **protected**: có thể truy cập được từ lớp khai báo và những lớp dẫn xuất từ nó.
- **private**: chỉ được truy cập bên trong bản thân lớp khai báo.

- **static**: phương thức lớp dùng chung cho tất cả các thể hiện của lớp, có nghĩa là phương thức đó có thể được thực hiện kể cả khi không có đối tượng của lớp chứa phương thức đó.
- **final**: phương thức có tiền tố này không được khai báo chồng ớ các lớp dẫn xuất.
- **abstract**: phương thức không cần cài đặt (không có phần source code), sẽ được hiện thực trong các lớp dẫn xuất từ lớp này.
- **synchoronized**: dùng để ngăn các tác động của các đối tượng khác lên đối tượng đang xét trong khi đang đồng bộ hóa. Dùng trong lập trình miltithreads.

<kiểu trả về>: có thể là kiểu void, kiểu cơ sở hay một lớp.
<Tên phương thức>: đặt theo qui ước giống tên biến.
<danh sách thông số>: có thể rỗng

Lưu ý:

Thông thường trong một lớp các phương thức nên được khai báo dùng từ khóa public, khác với vùng dữ liệu thường là dùng tiền tố private vì mục đích an toàn.

Những biến nằm trong một phương thức của lớp là các biến cục bộ (local) và nên được khởia tạo sau khi khai báo.

Ví dụ:

```
public class xemay
{
        public          String  nhasx;
        public          String  model;
        private         float   chiphisx;
        protected       int     thoigiansx;

        // so luong so cua xe may: 3, 4 so
        protected       int     so;
```

```
	// là biến tĩnh có giá trị là 2 trong tất cả
	// các thể hiện tạo ra từ lớp xemay
	public static    int sobanhxe = 2;

	public float tinhgiaban()
	{
		return 1.5 * chiphisx;
	}
}
```

3.2.6.Khởi tạo một đối tượng (Constructor)

Contructor thật ra là một loại phương thức đặc biệt của lớp. Constructor dùng gọi tự động khi khởi tạo một thể hiện của lớp, có thể dùng để khởi gán những giá trị măc định. Các constructor không có giá trị trả về, và có thể có tham số hoặc không có tham số.

Constructor phải có **cùng tên với lớp** và được gọi đến dùng từ khóa **new**.

Nếu một lớp không có constructor thì java sẽ cung cấp cho lớp một constructor mặc định (default constructor). Những thuộc tính, biến của lớp sẽ được khởi tạo bởi các giá trị mặc định (số: thường là giá trị 0, kiểu luận lý là giá trị false, kiểu đối tượng giá trị null, …)

Lưu ý: thông thường để an toàn, dễ kiểm soát và làm chủ mã nguồn chương trình chúng ta nên khai báo một constructor cho lớp.

Ví dụ:

```
public class xemay
{
	// …
	public xemay()
```

```
        {}
        public xemay(String s_nhasx, String s_model,
        f_chiphisx, int i_thoigiansx, int i_so);
        {
                nhasx = s_nhasx;
                model = s_model;
                chiphisx = f_chiphisx;
                thoigiansx = i_thoigiansx;
                so = i_so;

                // hoặc
                // this.nhasx = s_nhasx;
                // this.model = s_model;
                // this.chiphisx = f_chiphisx;
                // this.thoigiansx = i_thoigiansx;
                // this.so = i_so;
        }
}
```

3.2.7.Biến this

Biến **this** là một biến ẩn tồn tại trong tất cả các lớp trong ngông ngữ java. Một class trong Java luôn tồn tại một biến this, biến this được sử dụng trong khi chạy và tham khảo đến bản thân lớp chứa nó.

Ví dụ:

```
<tiền tố> class A
{
    <tiền tố> int <field_1>;
    <tiền tố> String <field_2>;

    // Contructor của lớp A
    public A(int par_1, String par_2)
    {
```

```
        this.field_1 = par_1;
        this.field_2 = par_2;
    }

    <tiền tố> <kiểu trả về> <method_1>()
    {
            // …
    }
    <tiền tố> <kiểu trả về> <method_2>()
    {
            this.method_1()
            // …
    }
}
```

3.2.8.Khai báo chồng phương thức (overloading method)

Việc khai báo trong một lớp nhiều phương thức có cùng tên nhưng khác tham số (khác kiểu dữ liệu, khác số lượng tham số) gọi là khai báo chồng phương thức (overloading method).

Ví dụ:

```
public class xemay
{       // khai báo fields …
        public float tinhgiaban()
        {       return 2 * chiphisx;
        }
        public float tinhgiaban(float huehong)
        {       return (2 * chiphisx + huehong);
        }
}
```

3.3.Đặc điểm hướng đối tượng trong java

Hỗ trợ những nguyên tắc cơ bản của lập trình hướng đối tượng, tất cả các ngôn ngữ lập trình kể cả java đều có ba đặc

điểm chung: tính đóng gói (encapsulation), tính đa hình (polymorphism), và tính kế thừa (inheritance).

3.3.1.Đóng gói (encapsulation)

Cơ chế đóng gói trong lập trình hướng đối tượng giúp cho các đối tượng dấu đi một phần các chi tiết cài đặt, cũng như phần dữ liệu cục bộ của nó, và chỉ công bố ra ngoài những gì cần công bố để trao đổi với các đối tượng khác. Hay chúng ta có thể nói đối tượng là một thành tố hỗ trợ tính đóng gói.

Đơn vị đóng gói cơ bản của ngôn ngữ java là class. Một class định nghĩa hình thức của một đối tượng. Một class định rõ những thành phần dữ liệu và các đoạn mã cài đặt các thao tác xử lý trên các đối tượng dữ liệu đó. Java dùng class để xây dựng những đối tượng. Những đối tượng là những thể hiện (instances) của một class.

Một lớp bao gồm thành phần dữ liệu và thành phần xử lý. Thành phần dữ liệu của một lớp thường bao gồm các biến thành viên và các biến thể hiện của lớp. Thành phần xử lý là các thao tác trên các thành phần dữ liệu, thường trong java người gọi là phương thức. Phương thức là một thuật ngữ hướng đối tượng trong java, trong C/C++ người ta thường dùng thuật ngữ là hàm.

3.3.2.Tính đa hình (polymorphism):

Tính đa hình cho phép cài đặt các lớp dẫn xuất khác nhau từ một lớp nguồn. Một đối tượng có thể có nhiều kiểu khác nhau gọi là tính đa hình.

Ví dụ:

```
class A_Object
{
        // …
        void method_1()
        {
                // …
```

```
        }
}

class B_Object extends A_Object
{
        // …
        void method_1()
        {
                // …
        }
}

class C
{       public static void main(String[] args)
        {
                // Tạo một mảng 2 phần tử kiểu A
                A_Object arr_Object = new A_Object[2];
                B_Object var_1 = new B_Object();
                // Phần tử đầu tiên của mảng arr_Object[0]
                tham // chiếu đến 1 đối tượng kiểu B_Object dẫn
                xuất // từ A_Object
                arr_Object[0] = var_1;
                A_Object var_2;
                for (int i=0; i<2; i++)
                {
                        var_2 = arr_Object[i];
                        var_2.method_1();
                }

        }
}
```

Vòng lặp for trong đoạn chương trình trên:

- Với i = 0 thì biến var_2 có kiểu là B_Object, và lệnh var_2.method_1() sẽ gọi thực hiện phương thức method_1 của lớp B_Object.
- Với i = 1 thì biến var_2 có kiểu là A_Object, và lệnh var_2.method_1() sẽ gọi thực hiện phương thức method_1 của lớp A_Object.

Trong ví dụ trên đối tượng var_2 có thể nhận kiểu A_Object hay B_Object. Hay nói các khác, một biến đối tượng kiểu A_Object như var_2 trong ví dụ trên có thể tham chiếu đến bất kỳ đối tượng nào của bất kỳ lớp con nào của lớp A_Object (ví dụ var_2 có thể tham chiếu đến đối tượng var_1, var_1 là đối tượng của lớp B_Object dẫn xuất từ lớp A_Object). Ngược lại một biến của lớp con không thể tham chiếu đến bất kỳ đối tượng nào của lớp cha.

3.3.3.Tính kế thừa (inheritance)

Một lớp con (subclass) có thể kế thừa tất cả những vùng dữ liệu và phương thức của một lớp khác (siêu lớp - superclass). Như vậy việc tạo một lớp mới từ một lớp đã biết sao cho các thành phần (fields và methods) của lớp cũ cũng sẽ thành các thành phần (fields và methods) của lớp mới. Khi đó ta gọi lớp mới là lớp dẫn xuất (derived class) từ lớp cũ (superclass). Có thể lớp cũ cũng là lớp được dẫn xuất từ một lớp nào đấy, nhưng đối với lớp mới vừa tạo thì lớp cũ đó là một lớp siêu lớp trực tiếp (immediate supperclass).

Dùng từ khóa **extends** để chỉ lớp dẫn xuất.

```
class A extends B
{
    // …
}
```

3.3.3.1 Khái báo phương thức chồng

Tính kế thừa giúp cho các lớp con nhận được các thuộc tính/phương thức public và protected của lớp cha. Đồng thời cũng có thể thay thế các phương thức của lớp cha bằng cách khai báo chồng. Chẳng hạn phương thức *tinhgiaban()* áp dụng trong lớp *xega* sẽ cho kết quả gấp 2.5 lần chi phí sản xuất thay vì gấp 2 chi phí sản xuất giống như trong lớp *xemay*.

Ví dụ:

```
public class xega extends xemay
{
        public xega()
        {
        }
        public xega(String s_nhasx, String s_model, f_chiphisx,
        int i_thoigiansx);
        {
                this.nhasx = s_nhasx;
                this.model = s_model;
                this.chiphisx = f_chiphisx;
                this.thoigiansx = i_thoigiansx;
                this.so = 0;
        }

        public float tinhgiaban()
        {
                return 2.5 * chiphisx;
        }
}
```

Java cung cấp 3 tiền tố/từ khóa để hỗ trợ tính kế thừa của lớp:

- **public**: lớp có thể truy cập từ các gói, chương trình khác.
- **final**: Lớp hằng, lớp không thể tạo dẫn xuất (không thể có con), hay đôi khi người ta gọi là lớp "vô sinh".

- **abstract**: Lớp trừu tượng (không có khai báo các thành phần và các phương thức trong lớp trừu tượng). Lớp dẫn xuất sẽ khai báo, cài đặt cụ thể các thuộc tính, phương thức của lớp trừu tượng.

3.3.3.2 Lớp nội

Lớp nội là lớp được khai báo bên trong 1 lớp khác. Lớp nội thể hiện tính đóng gói cao và có thể truy xuất trực tiếp biến của lớp cha.

Ví dụ:

```
public class A
{
	// …
	int <field_1>
	static class B
	{
		// …
		int <field_2>
		public B(int par_1)
		{
			field_2 = par_1 + field_1;
		}
	}
}
```

Trong ví dụ trên thì chương trình dịch sẽ tạo ra hai lớp với hai files khác nhau: A.class và B.class

3.3.3.3 Lớp vô sinh

Lớp không thể có lớp dẫn xuất từ nó (không có lớp con) gọi là lớp “vô sinh”, hay nói cách khác không thể kế thừa được từ một lớp “vô sinh”. Lớp “vô sinh” dùng để hạn chế, ngăn ngừa các lớp khác dẫn xuất từ nó.

Để khai báo một lớp là lớp "vô sinh", chúng ta dùng từ khóa final class.

Tất cả các phương thức của lớp vô sinh đều vô sinh, nhưng các thuộc tính của lớp vô sinh thì có thể không vô sinh.

Ví dụ:

```
public final class A
{
        public final int x;
        private int y;
        public final void method_1()
        {
                // …
        }
        public final void method_2()
        {
                // …
        }
}
```

3.3.3.4 Lớp trừu tượng

Lớp trừu tượng là lớp không có khai báo các thuộc tính thành phần và các phương thức. Các lớp dẫn xuất của nó sẽ khai báo thuộc tính, cài đặt cụ thể các phương thức của lớp trừu tượng.

Ví dụ:

```
abstract class A
{
    abstract void method_1();
```

```
}
public class B extends A
{
    public void method_1()
    {
        // cài đặt chi tiết cho phương thức method_1
        // trong lớp con B.
        // …
    }
}
public class C extends A
{
    public void method_1()
    {
        // cài đặt chi tiết cho phương thức method_1
        // trong lớp con C.
        // …
    }
}
```

Lưu ý: Các phương thức được khai báo dùng các tiền tố **private** và **static** thì không được khai báo là trừu tượng **abstract**. Tiền tố **private** thì không thể truy xuất từ các lớp dẫn xuất, còn tiền tố **static** thì chỉ dùng riêng cho lớp khai báo mà thôi.

3.3.3.5 Phương thức finalize()

Trong java không có kiểu dữ liệu con trỏ như trong C, người lập trình không cần phải quá bận tâm về việc cấp phát và giải phóng vùng nhớ, sẽ có một trình dọn dẹp hệ thống đảm trách việc này. Trình dọn dẹp hệ thống sẽ dọn dẹp vùng nhớ cấp phát cho các đối tượng trước khi hủy một đối tượng.

Phương thức *finalize()* là một phương thức đặc biệt được cài đặt sẵn cho các lớp. Trình dọn dẹp hệ thống sẽ gọi phương thức này trước khi hủy một đối tượng. Vì vậy việc cài đặt một số

thao tác giải phóng, dọn dẹp vùng nhớ đã cấp phát cho các đối tượng dữ liệu trong phương thức *finalize()* sẽ giúp cho người lập trình chủ động kiểm soát tốt quá trình hủy đối tượng thay vị giao cho trình dọn dẹp hệ thống tự động. Đồng thời việc cài đặt trong phương thức *finalize()* sẽ giúp cho bộ nhớ được giải phóng tốt hơn, góp phần cải tiến tốc độ chương trình.

Ví dụ:

```
class A
{
    // Khai báo các thuộc tính
    public void method_1()
    {
        // …
    }
    protected void finalize()
    {
        // Có thể dùng để đóng tất cả các kết nối
        // vào cơ sở dữ liệu trước khi hủy đối tượng.
        // …
    }
}
```

3.4.Gói (packages)

Việc đóng gói các lớp lại tạo thành một thư viện dùng chung gọi là package.

Một package có thể chứa một hay nhiều lớp bên trong, đồng thời cũng có thể chứa một package khác bên trong.

Để khai báo một lớp thuộc một gói nào đấy ta phải dùng từ khóa **package**.

Dòng khai báo gói phải là dòng đầu tiên trong tập tin khai báo lớp.

Các tập tin khai báo lớp trong cùng một gói phải được lưu trong cùng một thư mục.

Lưu ý: Việc khai báo import tất cả các lớp trong gói sẽ làm tốn bộ nhớ. Thông thường chúng ta chỉ nên import những lớp cần dùng trong chương trình.

Ví dụ:

```
package phuongtiengiaothong;
class xemay
{
        // ….
}
class xega extends xemay
{
        // …
}
```

Khi đó muốn sử dụng lớp *xemay* vào chương trình ta sẽ khai báo như sau:

```
        import phuongtiengiaothong.xemay;
```

3.5.Giao diện (interface)

3.5.1.Khái niệm interface:

Như chúng ta đã biết một lớp trong java chỉ có một siêu lớp trực tiếp hay một cha duy nhất (đơn thừa kế). Để tránh đi tính phức tạp của đa thừa kế (multi-inheritance) trong lập trình hướng đối tượng, Java thay thế bằng giao tiếp (interface). Một lớp có thể có nhiều giao tiếp (interface) với các lớp khác để

thừa hưởng thêm vùng dữ liệu và phương thức của các giao tiếp này.

3.5.2.Khai báo interface:

Interface được khai báo như một lớp. Nhưng các thuộc tính của interface là các hằng (khai báo dùng từ khóa final) và các phương thức của giao tiếp là trừu tượng (mặc dù không có từ khóa abstract).

Trong các lớp có cài đặt các interface ta phải tiến hành cài đặt cụ thể các phương thức này.

Ví dụ:

```
public interface sanpham
{       static  final   String  nhasx = "Honda VN";
        static  final   String  dienthoai = "08-8123456";
        public  int     gia(String s_model);
}

// khai báo 1 lớp có cài đặt interface
public class xemay implements sanpham
{       // cài đặt lại phương thức của giao diện trong lớp
        public  int     gia(String s_model)
        {
                if (s_model.equals("2005"))
                        return (2000);
                else
                        return (1500);

        }

        public  String chobietnhasx()
        {
                return (nhasx);
        }
}
```

Có một vấn đề khác với lớp là một giao diện (interface) không chỉ có một giao diện cha trực tiếp mà có thể dẫn xuất cùng lúc nhiều giao diện khác (hay có nhiều giao diện cha). Khi đó nó sẽ kế thừa tất cả các giá trị hằng và các phương thức của các giao diện cha. Các giao diện cha được liệt kê thành chuỗi và cách nhau bởi dấu phẩy ",". Khai báo như sau:

```
public interface InterfaceName extends interface1, interface2,
interface3
{
        // …
}
```

3.5.3.Ví dụ minh họa

Ví dụ 1: Minh họa tính đa hình (polymorphism) trong phân cấp kế thừa thông qua việc mô tả và xử lý một số thao tác cơ bản trên các đối tượng hình học.

```
// Định nghĩa lớp trừu tượng cơ sở tên Shape trong
// tập tin Shape.java
public abstract class Shape extends Object
{
    // trả về diện tích của một đối tượng hình học shape
    public double area()
    {
        return 0.0;
    }

    // trả về thể tích của một đối tượng hình học shape
    public double volume()
    {
        return 0.0;
    }
```

```
    // Phương thức trừu tượng cần phải được hiện thực
    // trong những lớp con để trả về tên đối tượng
    // hình học shape thích hợp
    public abstract String getName();
} // end class Shape

// Định nghĩa lớp Point trong tập tin Point.java
public class Point extends Shape
{
        protected int x, y; // Tọa độ x, y của 1 điểm

        // constructor không tham số.
        public Point()
        {
                setPoint( 0, 0 );
        }

        // constructor có tham số.
        public Point(int xCoordinate, int yCoordinate)
        {
                setPoint( xCoordinate, yCoordinate );
        }

        // gán tọa độ x, y cho 1 điểm
        public void setPoint( int xCoordinate, int yCoordinate )
        {
                x = xCoordinate;
                y = yCoordinate;
        }

        // lấy tọa độ x của 1 điểm
        public int getX()
        {
                return x;
        }
```

```
    // lấy tọa độ y của 1 điểm
    public int getY()
    {
            return y;
    }

    // Thể hiện tọa độ của 1 điểm dưới dạng chuỗi
    public String toString()
    {
            return "[" + x + ", " + y + "]";
    }

    // trả về tên của đối tượng shape
    public String getName()
    {
            return "Point";
    }

} // end class Point
```

Định nghĩa một lớp cha **Shape** là một lớp trừu tượng dẫn xuất từ Object và có 3 phương thức khai báo dùng tiền tố public. Phương thức **getName()** khai báo trừu tượng vì vậy nó phải được hiện thực trong các lớp con. Phương thức **area()** (tính diện tích) và phương thức **volume()** (tính thể tích) được định nghĩa và trả về 0.0. Những phương thức này sẽ được khai báo chồng trong các lớp con để thực hiện chức năng tính diện tích cũng như thể tích phù hợp với những đối tượng hình học tương ứng (đường tròn, hình trụ, …)

Lớp **Point:** dẫn xuất từ lớp **Shape**. Một điểm thì có diện tích và thể tích là 0.0, vì vậy những phương thức **area**() và **volume**() của lớp cha không cần khai báo chồng trong lớp **Point,** chúng được thừa kế như đã định nghĩa trong lớp trừu tượng **Shape**. Những phương thức khác như **setPoint**(…) để

gán tọa độ x, y cho một điểm, còn phương thức **getX()**, **getY()** trả về tọa độ x, y của một điểm. Phương thức **getName()** là hiện thực cho phương thức trừu tượng trong lớp cha, nếu như phương thức **getName()** mà không được định nghĩa thì lớp **Point** là một lớp trừu tượng.

```
// Định nghĩa lớp Circle trong tập tin Circle.java
public class Circle extends Point
{       // Dẫn xuất từ lớpPoint
        protected double radius;

        // constructor không tham số
        public Circle()
        {
                // ngầm gọi đến constructor của lớp cha
                setRadius( 0 );
        }

        // constructor có tham số
        public Circle( double circleRadius, int xCoordinate,
        int yCoordinate )
        {
                // gọi constructorcủa lớp cha
                super( xCoordinate, yCoordinate );

                setRadius( circleRadius );
        }

        // Gán bán kính của đường tròn
        public void setRadius( double circleRadius )
        {
                radius = ( circleRadius >= 0 ? circleRadius:0 );
        }

        // Lấy bán kính của đường tròn
```

```
        public double getRadius()
        {
                return radius;
        }

        // Tính diện tích đường tròn Circle
        public double area()
        {
                return Math.PI * radius * radius;
        }

        // Biểu diễn đường tròn bằng một chuỗi
        public String toString()
        {
                return "Center = " + super.toString() +
                "; Radius = " + radius;
        }

        // trả về tên của shape
        public String getName()
        {
                return "Circle";
        }
} // end class Circle
```

Lớp **Circle** dẫn xuất từ lớp **Point**, một đường tròn có thể tích là 0.0, vì vậy phương thức **volume()** của lớp cha không khai báo chồng, nó sẽ thừa kế từ lớp **Point**, mà lớp **Point** thì thừa kế từ lớp **Shape**. Diện tích đường tròn khác với một điểm, vì vậy phương thức tính diện tích **area()** được khai báo chồng. Phương thức **getName()** hiện thực phương thức trừu tượng đã khai báo trong lớp cha, nếu phương thức **getName()** không khai báo trong lớp **Circle** thì nó sẽ kế thừa từ lớp **Point**. Phương thức **setRadius** dùng để gán một bán kính (radius) mới cho một

đối tượng đường tròn, còn phương thức **getRadius** trả về bán kính của một đối tượng đường tròn.

```
// Định nghĩa lớp hình trụ Cylinder
// trong tập tin Cylinder.java.
public class Cylinder extends Circle
{
	// chiều cao của Cylinder
	protected double height;

	// constructor không có tham số
	public Cylinder()
	{
		// ngầm gọi đến constructor của lớp cha
		setHeight( 0 );
	}

	// constructor có tham số
	public Cylinder( double cylinderHeight,
	double cylinderRadius, int xCoordinate,
	int yCoordinate )
	{
		// Gọi constructor của lớp cha
		super( cylinderRadius, xCoordinate,
		yCoordinate );

		setHeight( cylinderHeight );
	}

	// Gán chiều cao cho Cylinder
	public void setHeight( double cylinderHeight )
	{
		height = ( cylinderHeight >= 0 ? cylinderHeight
		:0 );
	}
```

```
 // Lấy chiều cao của Cylinder
public double getHeight()
{
        return height;
}

// Tính diện tích xung quanh của Cylinder
public double area()
{
        return 2 * super.area() + 2 * Math.PI * radius *
        height;
}

// Tính thể tích của Cylinder
public double volume()
{
        return super.area() * height;
}

// Biểu diễn Cylinder bằng một chuỗi
public String toString()
{
        return super.toString() + "; Height = " + height;
}

// trả về tên của shape
public String getName()
{
        return "Cylinder";
}

} // end class Cylinder
```

Lớp **Cylinder** dẫn xuất từ lớp Circle. Một Cylinder (hình trụ) có diện tích và thể tích khác với một Circle (hình tròn), vì vậy cả hai phương thức **area()** và **volume()** cần phải khai báo chồng. Phương thức getName() là hiện thực phương thức trừu tượng trong lớp cha, nếu phương thức **getName()** không khai báo trong lớp **Cylinder** thì nó sẽ kế thừa từ lớp **Circle**. Phương thức **setHeight** dùng để gán chiều cao mới cho một đối tượng hình trụ, còn phương thức **getHeight** trả về chiều cao của một đối tượng hình trụ.

```
// Test.java
// Kiểm tra tính kế thừa của Point, Circle, Cylinder với
// lớp trừu tượng Shape.
// Khai báo thư viện
import java.text.DecimalFormat;
public class Test
{
    // Kiểm tra tính kế thừa của các đối tượng hình học
    public static void main( String args[] )
    {
        // Tạo ra các đối tượng hìnhhọc
        Point point = new Point( 7, 11 );
        Circle circle = new Circle( 3.5, 22, 8 );
        Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );

        // Tạo một mảng các đối tượng hình học
        Shape arrayOfShapes[] = new Shape[ 3 ];

        // arrayOfShapes[ 0 ] là một đối tượng Point
        arrayOfShapes[ 0 ] = point;
        // arrayOfShapes[ 1 ] là một đối tượng Circle
        arrayOfShapes[ 1 ] = circle;
        // arrayOfShapes[ 2 ] là một đối tượng cylinder
        arrayOfShapes[ 2 ] = cylinder;

        // Lấy tên và biểu diễn của mỗi đối tượng hình học
```

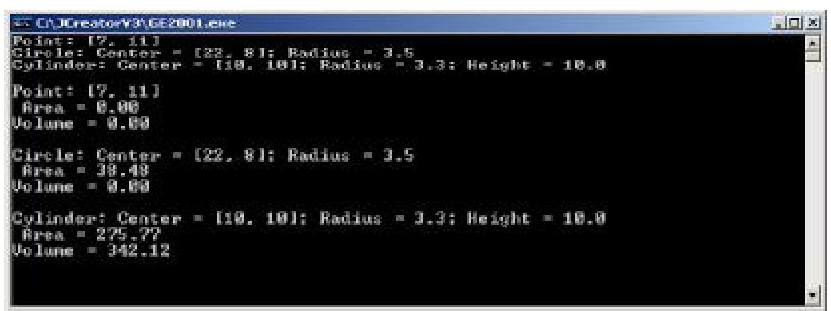
```
        String output =
        point.getName() + ": " + point.toString() + "\n" +
        circle.getName() + ": " + circle.toString() + "\n" +
        cylinder.getName() + ": " + cylinder.toString();

        DecimalFormat precision2 = new DecimalFormat(
        "0.00" );

        // duyệt mảng arrayOfShapes lấy tên, diện tích, thể tích
        // của mỗi đối tượng hình học trong mảng.
        for ( int i = 0; i < arrayOfShapes.length; i++ )
        {
            output += "\n\n" + arrayOfShapes[ i ].getName() +
            ": " + arrayOfShapes[ i].toString() +
            "\n Area = " +
            precision2.format( arrayOfShapes[ i ].area() ) +
            "\nVolume = " +
            precision2.format( arrayOfShapes[ i ].volume() );
        }

        System.out.println(output);
        System.exit( 0 );
    }
} // end class Test
```

Kết quả thực thi chương trình:



Ví dụ 2: Tương tự ví dụ 1 nhưng trong ví dụ 2 chúng ta dùng interface để định nghĩa cho **Shape** thay vì một lớp trừu tượng. Vì vậy tất cả các phương thức trong interface **Shape** phải được hiện thực trong lớp **Point** là lớp cài đặt trực tiếp interface **Shape**.

```
// Định nghĩa một interface Shape trong tập tin shape.java
public interface Shape
{
        // Tính diện tích
        public abstract double area();

        // Tính thể tích
        public abstract double volume();

        // trả về tên của shape
        public abstract String getName();
}
```

Lớp *Point* cài đặt/hiện thực *interface* tên *shape*.

```
// Định nghĩa lớp Point trong tập tin Point.java
public class Point extends Object implements Shape
{
        protected int x, y; // Tọa độ x, y của 1 điểm

        // constructor không tham số.
        public Point()
        {
                setPoint( 0, 0 );
        }

        // constructor có tham số.
        public Point(int xCoordinate, int yCoordinate)
        {
                setPoint( xCoordinate, yCoordinate );
```

```
}

// gán tọa độ x, y cho 1 điểm
public void setPoint( int xCoordinate, int yCoordinate )
{
        x = xCoordinate;
        y = yCoordinate;
}

// lấy tọa độ x của 1 điểm
public int getX()
{
        return x;
}

// lấy tọa độ y của 1 điểm
public int getY()
{
        return y;
}

// Thể hiện tọa độ của 1 điểm dưới dạng chuỗi
public String toString()
{
        return "[" + x + ", " + y + "]";
}

// Tính diện tích
public double area()
{
        return 0.0;
}

// Tính thể tích
public double volume()
```

```
        {
                return 0.0;
        }

        // trả về tên của đối tượng shape
        public String getName()
        {
                return "Point";
        }

} // end class Point
```

Lớp *Circle* là lớp con của lớp *Point*, và cài đặt/hiện thực gián tiếp *interface* tên *shape*.

```
// Định nghĩa lớp Circle trong tập tin Circle.java
public class Circle extends Point
{       // Dẫn xuất từ lớpPoint
        protected double radius;

        // constructor không tham số
        public Circle()
        {
                // ngầm gọi đến constructor của lớp cha
                setRadius( 0 );
        }

        // constructor có tham số
        public Circle( double circleRadius, int xCoordinate,
        int yCoordinate )
        {
                // gọi constructorcủa lớp cha
                super( xCoordinate, yCoordinate );

                setRadius( circleRadius );
        }
```

```
        // Gán bán kính của đường tròn
        public void setRadius( double circleRadius )
        {
                radius = ( circleRadius >= 0 ? circleRadius:0 );
        }

        // Lấy bán kính của đường tròn
        public double getRadius()
        {
                return radius;
        }

        // Tính diện tích đường tròn Circle
        public double area()
        {
                return Math.PI * radius * radius;
        }

        // Biểu diễn đường tròn bằng một chuỗi
        public String toString()
        {
                return "Center = " + super.toString() +
                "; Radius = " + radius;
        }

        // trả về tên của shape
        public String getName()
        {
                return "Circle";
        }
} // end class Circle

// Định nghĩa lớp hình trụ Cylinder
// trong tập tin Cylinder.java.
```

```
public class Cylinder extends Circle
{
	// chiều cao của Cylinder
	protected double height;

	// constructor không có tham số
	public Cylinder()
	{
		// ngầm gọi đến constructor của lớp cha
		setHeight( 0 );
	}

	// constructor có tham số
	public Cylinder( double cylinderHeight,
	double cylinderRadius, int xCoordinate,
	int yCoordinate )
	{
		// Gọi constructor của lớp cha
		super( cylinderRadius, xCoordinate,
		yCoordinate );

		setHeight( cylinderHeight );
	}

	// Gán chiều cao cho Cylinder
	public void setHeight( double cylinderHeight )
	{
		height = ( cylinderHeight >= 0 ? cylinderHeight
		:0 );
	}

	 // Lấy chiều cao của Cylinder
	public double getHeight()
	{
		return height;
```

```
    }

    // Tính diện tích xung quanh của Cylinder
    public double area()
    {
        return 2 * super.area() + 2 * Math.PI * radius *
        height;
    }

    // Tính thể tích của Cylinder
    public double volume()
    {
        return super.area() * height;
    }

    // Biểu diễn Cylinder bằng một chuỗi
    public String toString()
    {
        return super.toString() + "; Height = " + height;
    }

    // trả về tên của shape
    public String getName()
    {
        return "Cylinder";
    }

} // end class Cylinder

// Test.java
// Kiểm tra tính kế thừa của Point, Circle, Cylinder với
// interface Shape.

// Khai báo thư viện
import java.text.DecimalFormat;
```

```
public class Test
{
    // Kiểm tra tính kế thừa của các đối tượng hình học
    public static void main( String args[] )
    {
        // Tạo ra các đối tượng hìnhhọc
        Point point = new Point( 7, 11 );
        Circle circle = new Circle( 3.5, 22, 8 );
        Cylinder cylinder = new Cylinder( 10, 3.3, 10, 10 );

        // Tạo một mảng các đối tượng hình học
        Shape arrayOfShapes[] = new Shape[ 3 ];

        // arrayOfShapes[ 0 ] là một đối tượng Point
        arrayOfShapes[ 0 ] = point;
        // arrayOfShapes[ 1 ] là một đối tượng Circle
        arrayOfShapes[ 1 ] = circle;
        // arrayOfShapes[ 2 ] là một đối tượng cylinder
        arrayOfShapes[ 2 ] = cylinder;

        // Lấy tên và biểu diễn của mỗi đối tượng hình học
        String output =
        point.getName() + ": " + point.toString() + "\n" +
        circle.getName() + ": " + circle.toString() + "\n" +
        cylinder.getName() + ": " + cylinder.toString();

        DecimalFormat precision2 = new DecimalFormat(
        "0.00" );

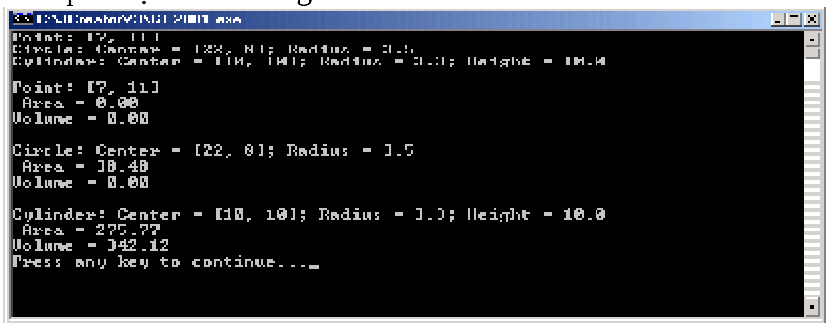
        // duyệt mảng arrayOfShapes lấy tên, diện tích, thể tích
        // của mỗi đối tượng hình học trong mảng.
        for ( int i = 0; i < arrayOfShapes.length; i++ )
        {
            output += "\n\n" + arrayOfShapes[ i ].getName() +
            ": " + arrayOfShapes[ i].toString() +
```

```
            "\n Area = " +
            precision2.format( arrayOfShapes[ i ].area() ) +
            "\nVolume = " +
            precision2.format( arrayOfShapes[ i ].volume() );
        }

        System.out.println(output);
        System.exit( 0 );
    }

} // end class Test
```

Kết quả thực thi chương trình:



Chương 4: THIẾT KẾ GIAO DIỆN NGƯỜI DÙNG

4.1.Mở đầu

Chương này cung cấp cho sinh viên những kiến thức cơ bản để xây dựng giao diện (Graphic User Interface - GUI) của chương trình ứng dụng bằng ngôn ngữ java:

- Những nguyên tắc thiết kế giao diện.
- Những thư viện, gói xây dựng giao diện: gồm những lớp (class), những giao tiếp (interface) quản lý sự kiện và những thành phần (components) xây dựng nên giao diện người dùng.
- Bộ quản lý trình bày (layout managers)
- Xử lý sự kiện

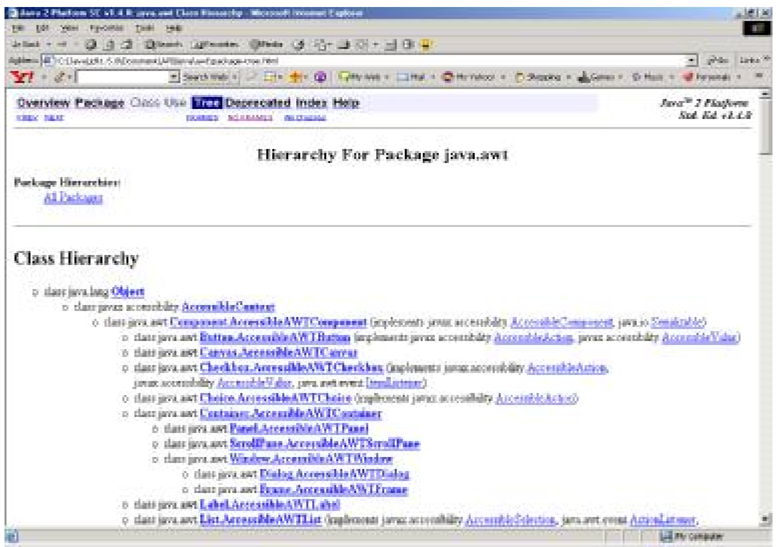
Trong khuôn khổ giáo trình lập trình java căn bản này chúng tôi trình bày việc thiết kế GUI dùng thư viện awt (abstract windows toolkit). Việc thiết kết GUI sẽ trực quan, uyển chuyển hơn khi chúng ta sử dụng thư viện JFC (Java Foundation Class) sẽ giới được giới thiệu trong chuyên đề java nâng cao.

4.2.Giới thiệu thư viện awt

Thư viện awt là bộ thư viện dùng để xây dựng giao diện người dùng cho một chương trình ứng dụng có đầy đủ các thành phần cơ bản như: Label, Button, Checkbox, Radiobutton, Choice, List, Text Field, Text Area, Scrollbar, Menu, Frame…

Giống như các API của Windows, java cung cấp cho người lập trình thư viện awt. Nhưng khác với các hàm API, thư viện awt không phụ thuộc hệ điều hành. Thư viện awt là nền tảng, cơ sở giúp cho chúng ta tiếp cận với thư viện mở rộng JFC hiệu quả hơn.

Cấu trúc cây phân cấp của tất cả những lớp trong thư viện awt chúng ta có thể xem chi tiết trong tài liệu kèm theo bộ công cụ j2se (phần API Specification)



4.3.Các khái niệm cơ bản

4.3.1.Component

Component là một đối tượng có biểu diễn đồ họa được hiển thị trên màn hình mà người dùng có thể tương tác được. Chẳng

hạn như những nút nhấn (button), những checkbox, những scrollbar,… Lớp **Component** là một lớp trừu tượng.

java.lang.Object
└ **java.awt.Component**

4.3.2.Container

Container là đối tượng vật chứa hay những đối tượng có khả năng quản lý và nhóm các đối tượng khác lại. Những đối tượng con thuộc thành phần awt như: button, checkbox, radio button, scrollbar, list,… chỉ sử dụng được khi ta đưa nó vào khung chứa (container).

Một số đối tượng container trong Java:

- **Panel**: Đối tượng khung chứa đơn giản nhất, dùng để nhóm các đối tượng, thành phần con lại. Một Panel có thể chứa bên trong một Panel khác.
  java.lang.Object
    +--java.awt.Component
        +--java.awt.Container
            +--**java.awt.Panel**
- **Frame**: khung chứa Frame là một cửa số window hẳn hoi ở mức trên cùng bao gồm một tiêu đều và một đường biên (border) như các ứng dụng windows thông thường khác. Khung chứa Frame thường được sử dụng để tạo ra cửa sổ chính của các ứng dụng.
  java.lang.Object
    +--java.awt.Component
        +--java.awt.Container
            +--java.awt.Window
                +--**java.awt.Frame**
- **Dialogs**: đây là một cửa sổ dạng hộp hội thoại (cửa sổ dạng này còn được gọi là pop-up window), cửa sổ dạng này thường được dùng để đưa ra thông báo, hay dùng để lấy dữ liệu nhập từ ngoài vào thông qua các đối tượng, thành phần trên dialog như TextField chẳng hạn. Dialog

cũng là một cửa sổ nhưng không đầy đủ chức năng như đối tượng khung chứa Frame.

```
java.lang.Object
  +--java.awt.Component
        +--java.awt.Container
              +--java.awt.Window
                    +--java.awt.Dialog
```

- **ScrollPanes**: là một khung chứa tương tự khung chứa Panel, nhưng có thêm 2 thanh trượt giúp ta tổ chức và xem được các đối tượng lớn choán nhiều chỗ trên màn hình như những hình ảnh hay văn bản nhiều dòng.

```
java.lang.Object
  +--java.awt.Component
        +--java.awt.Container
              +--java.awt.ScrollPane
```

4.3.3.Layout Manager

Khung chứa container nhận các đối tượng từ bên ngoài đưa vào và nó phải biết làm thế nào để tổ chức sắp xếp "chỗ ở" cho các đối tượng đó. Mỗi đối tượng khung chứa đều có một bộ quản lý chịu trách nhiệm thực hiện công việc đấy đó là bộ quản lý trình bày (Layout Manager). Các bộ quản lý trình bày mà thư viện AWT cung cấp cho ta bao gồm:

- **FlowLayout**: Sắp xếp các đối tượng ***từ trái qua phải và từ trên xuống dưới***. Các đối tượng đều giữ nguyên kích thước của mình.
- **BorderLayout**: Các đối tượng được đặt theo các đường viền của khung chứa theo các cạnh ***West, East, South, North*** và ***Center*** tức Đông, Tây, Nam, Bắc và Trung tâm hay Trái, Phải, Trên, Dưới và Giữa tùy theo cách nhìn của chúng ta.
- **GridLayout**: Tạo một khung lưới vô hình với các ô bằng nhau. Các đối tượng sẽ đặt vừa kích thước với

từng ô đó. Thứ tự sắp xếp cũng từ trái qua phải và từ trên xuống dưới.

- **GridBagLayout**: Tương tự như GridLayout, các đối tượng khung chứa cũng được đưa vào một lưới vô hình. Tuy nhiên kích thước các đối tượng không nhất thiết phải vừa với 1 ô mà có thể là 2, 3 ô hay nhiều hơn tùy theo các ràng buộc mà ta chỉ định thông qua đối tượng GridBagConstraint.
- **Null Layout**: Cách trình bày tự do. Đối với cách trình bày này người lập trình phải tự động làm tất cả từ việc định kích thước của các đối tượng, cũng như xác định vị trí của nó trên màn hình. Ta không phụ thuộc vào những ràng buộc đông, tây , nam, bắc gì cả.

4.4.Thiết kế GUI cho chương trình

4.4.1.Tạo khung chứa cửa sổ chương trình

Thông thường để tạo cửa sổ chính cho chương trình ứng dụng ta tiến hành các bước:

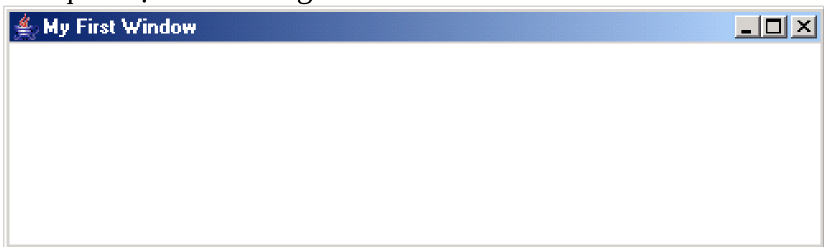
- Tạo đối tượng Frame
- Xác định kích thước của Frame
- Thể hiện Frame trên màn hình

Ví dụ:

```
import java.awt.*;
class FrameDemo
{
    public static void main(String args[])
    {
            // Tạo đối tượng khung chứaFrame
            Frame fr = new Frame("My First Window") ;
            // Xác định kích thước, vị trí của Frame
            fr.setBounds(0, 0, 640, 480);
            // Hiển thị Frame
```

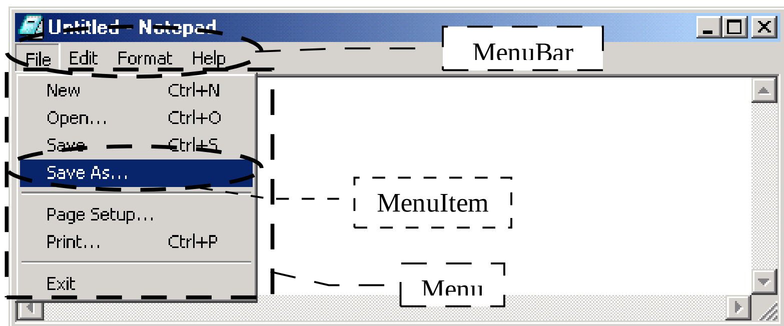
```
            fr.setVisible(true);
        }
    }
```

Kết quả thực thi chương trình:



4.4.2.Tạo hệ thống thực đơn

Đối với thư viện awt, để xây dựng hệ thống thực đơn cho chương trình ứng dụng chúng ta có thể dùng các lớp MenuBar, Menu, MenuItem, MenuShortcut.



Ví dụ: Tạo hệ thống thực đơn cho chương trình Calculator

```
import java.awt.*;
import java.awt.event.*;
class Calculator
{
    public static void main(String[] args)
    {
```

```
    createMenu();
}

private static void createMenu()
{
    // Tao Frame ung dung
    final Frame fr = new Frame();
    fr.setLayout(new BorderLayout());

    // Tao cac menu bar
    MenuBar menu = new MenuBar();
    Menu menuFile = new Menu("Edit");
    MenuItem copyItem = new MenuItem("Copy Ctrl+C");
    MenuItem pasteItem = new MenuItem("Paste Ctrl+V");
    menuFile.add(copyItem);
    menuFile.add(pasteItem);

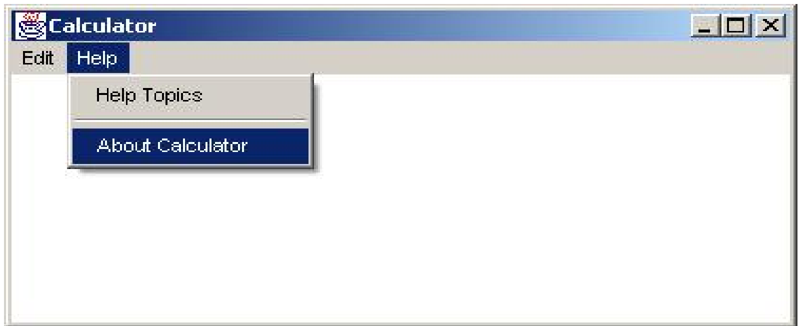
    Menu menuHelp = new Menu("Help");
    MenuItem hTopicItem = new MenuItem("Help Topics");
    MenuItem hAboutItem = new MenuItem("About
    Calculator");
    menuHelp.add(hTopicItem);
    menuHelp.addSeparator();
    menuHelp.add(hAboutItem);
    menu.add(menuFile);
    menu.add(menuHelp);

    fr.setMenuBar(menu);
    fr.setBounds(100, 100, 300, 200);
    fr.setTitle("Calculator");
    //fr.setResizable(false);
    fr.setVisible(true);

    // xử lý biến sự kiện đóng cửa số ứng dụng.
    fr.addWindowListener(
```

```
            new WindowAdapter()
            {
                public void windowClosing(WindowEvent e)
                {
                    System.exit(0);
                }
            });
    }
}
```

Kết quả thực thi chương trình:



4.4.3.Gắn Component vào khung chứa

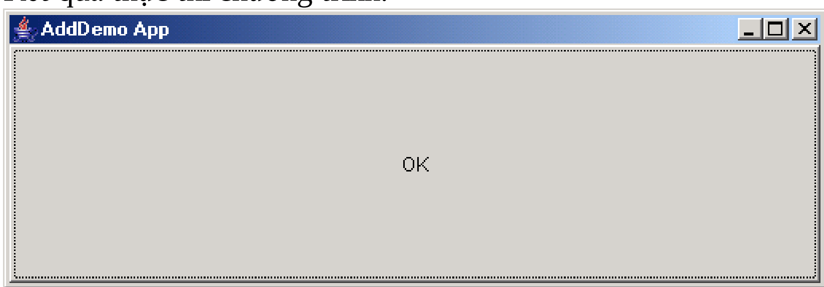
Để gắn một thành phần, một đối tượng component vào một cửa số (khung chứa) chúng ta dùng phương thức add của đối tượng khung chứa container.

Ví dụ:

```
import java.awt.*;
class AddDemo
{
    public static void main(String args[])
    {
            // Tạo đối tượng khung chứaFrame
            Frame fr = new Frame("AddDemo App");
            // Tạo đối tượng Component
```

```
            Button buttOk = new Button("OK");
            // Gắn đối tượng nút nhấn vào khung chứa
            fr.add(buttOk);
            // Xác định kích thước, vị trí của Frame
            fr.setSize(100, 100);
            // Hiển thị Frame
            fr.setVisible(true);
        }
    }
```

Kết quả thực thi chương trình:



4.4.4.Trình bày các Component trong khung chứa

Như chúng ta đã biết khung chứa container nhận các đối tượng từ bên ngoài đưa vào và nó phải biết làm thế nào để tổ chức sắp xếp "chỗ ở" cho các đối tượng đó. Mỗi đối tượng khung chứa đều có một bộ quản lý chịu trách nhiệm thực hiện công việc đấy đó là bộ quản lý trình bày (Layout Manager). Chúng ta sẽ tìm hiểu chi tiết về các kiểu trình bày của thư viện AWT.

Interface LayoutManager định nghĩa giao tiếp cho những lớp biết được làm thế nào để trình bày những trong những containers

4.4.4.1 FlowLayout

public class **FlowLayout** extends Object

implements LayoutManager, Serializable

Đối với một container trình bày theo kiểu FlowLayout thì:

- Các component gắn vào được sắp xếp theo thứ tự từ trái sang phải và từ trên xuống dưới.
- Các component có kích thước như mong muốn.
- Nếu chiều rộng của Container không đủ chỗ cho các component thì chúng tự động tạo ra một dòng mới.
- FlowLayout thường được dùng để để sắp xếp các button trong 1 panel.
- Chúng ta có thể điều chỉnh khoảng cách giữa các component.

Ví dụ:

```
import java.awt.*;
import java.lang.Integer;
class FlowLayoutDemo
{
    public static void main(String args[])
    {
        Frame fr = new Frame("FlowLayout Demo");
        fr.setLayout(new FlowLayout());
        fr.add(new Button("Red"));
        fr.add(new Button("Green"));
        fr.add(new Button("Blue"));

        List li = new List();
        for (int i=0; i<5; i++)
        {
            li.add(Integer.toString(i));
        }

        fr.add(li);
        fr.add(new Checkbox("Pick me", true));
        fr.add(new Label("Enter your name:"));
```

```
        fr.add(new TextField(20));
        // phương thức pack() được gọi sẽ làm cho cửa sổ
        // hiện hành sẽ có kích thước vừa với kích thước
        // trình bày bố trí những thành phần con của nó.
        fr.pack();
        fr.setVisible(true);
    }
}
```

Kết quả thực thi chương trình:



4.4.4.2 BorderLayout

public class **BorderLayout** extends Object
implements LayoutManager2, Serializable

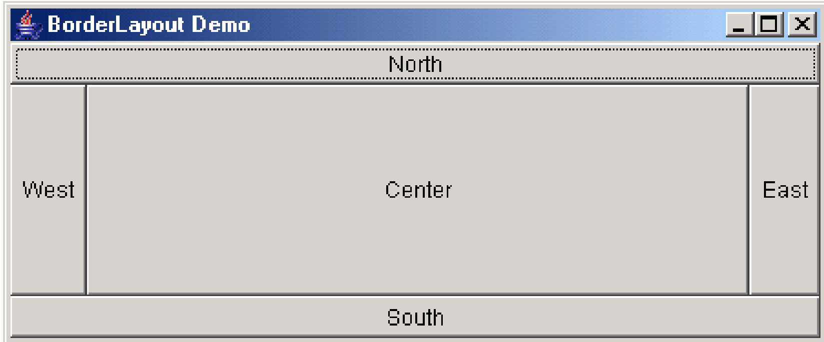
Đối với một container trình bày theo kiểu BorderLayout thì:

- Bộ trình bày khung chứa được chia làm 4 vùng: NORTH, SOUTH, WEST, EAST và CENTER. (Đông, Tây, Nam, Bắc và trung tâm). Bộ trình bày loại này cho phép sắp xếp và thay đổi kích thước của những components chứa trong nó sao cho vứa với 5 vùng ĐÔNG, TÂY, NAM, BẮC, TRUNG TÂM.
- Không cần phải gắn component vào cho tất cả các vùng.
- Các component ở vùng NORTH và SOUTH có chiều cao tùy ý nhưng có chiều rộng đúng bằng chiều rộng vùng chứa.
- Các component ở vùng EAST và WEST có chiều rộng tùy ý nhưng có chiều cao đúng bằng chiều cao vùng chứa.
- Các component ở vùng CENTER có chiều cao và chiều rộng phụ thuộc vào các vùng xung quanh.

Ví dụ:

```
import java.awt.*;
class BorderLayoutDemo extends Frame
{
    private Button north, south, east, west, center;
    public BorderLayoutDemo(String sTitle)
    {
        super(sTitle);
        north   = new Button("North");
        south   = new Button("South");
        east    = new Button("East");
        west    = new Button("West");
        center  = new Button("Center");
        this.add(north, BorderLayout.NORTH);
        this.add(south, BorderLayout.SOUTH);
        this.add(east, BorderLayout.EAST);
        this.add(west, BorderLayout.WEST);
        this.add(center, BorderLayout.CENTER);
    }
    public static void main(String args[])
    {
        Frame fr = new BorderLayoutDemo ("BorderLayout
        Demo");
        fr.pack();
        fr.setVisible(true);
    }
}
```

Kết quả thực thi chương trình:



4.4.4.3 GridLayout

public class **GridLayout** extends Object
implements LayoutManager

Đối với một container trình bày theo kiểu GridLayout thì:

- Bộ trình bày tạo một khung lưới vô hình với các ô bằng nhau.
- Các đối tượng sẽ đặt vừa kích thước với từng ô đó. Thứ tự sắp xếp từ trái qua phải và từ trên xuống dưới.

Ví dụ:

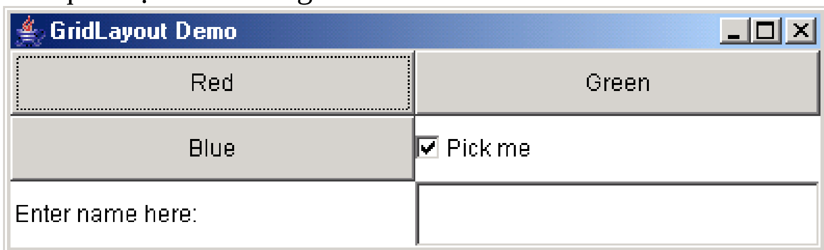
```
import java.awt.*;
public class GridLayoutDemo
{
	public static void main(String arg[])
	{
		Frame f = new Frame("GridLayout Demo");
		f.setLayout(new GridLayout(3,2));

		f.add(new Button("Red"));
		f.add(new Button("Green"));
		f.add(new Button("Blue"));
		f.add(new Checkbox("Pick me", true));
		f.add(new Label("Enter name here:"));
```

```
            f.add(new TextField());

            f.pack();
            f.setVisible(true);
      }
}
```

Kết quả thực thi chương trình:



4.4.4.4 GridBagLayout

public class **GridBagLayout** extends Object
implements LayoutManager2
(public interface **LayoutManager2** extends LayoutManager)

Đối với một container trình bày theo kiểu GridBagLayout thì:

- Các componets khi được đưa vào khung chứa sẽ được trình bày trên 1 khung lưới vô hình tương tự như GridLayout. Tuy nhiên khác với GridLayout kích thước các đối tượng không nhất thiết phải vừa với 1 ô trên khung lưới mà có thể là 2, 3 ô hay nhiều hơn tùy theo các ràng buộc mà ta chỉ định thông qua đối tượng GridBagConstraints.
- Lớp **GridBagConstraints** dẫn xuất từ lớp Object. Lớp GridBagConstraints dùng để chỉ định ràng buộc cho những components trình bày trong khung chứa container theo kiểu GridBagLayout.
  - o **gridx, gridy**: vị trí ô của khung lưới vô hình mà ta sẽ đưa đối tượng con vào

- **gridwidth, gridheight**: kích thước hay vùng trình bày cho đối tượng con.
- **Insets**: là một biến đối tượng thuộc lớp Inset dùng để qui định khoảng cách biên phân cách theo 4 chiều (trên, dưới, trái, phải).
- **weightx, weighty**: chỉ định khoảng cách lớn ra tương đối của các đối tượng con với nhau

Ví dụ:

```
import java.awt.*;
public class GridBagLayoutDemo
{
    public static void main(String arg[])
    {
        Frame f = new Frame("GridBagLayout Demo");

        // Thiet lap layout manager
        // Tao doi tuong rang buoc cho cach trinh bay
        // GridBagLayout.
        GridBagLayout layout = new GridBagLayout();
        GridBagConstraints constraints = new
        GridBagConstraints();
        f.setLayout(layout);

        // Tao ra 9 nut nhan
        String[] buttName = {"Mot", "Hai", "Ba", "Bon",
        "Nam", "Sau", "Bay", "Tam", "Chin"};
        Button[] buttons = new Button[9];

        for(int i=0;i<9;i++)
        {
            buttons[i] = new Button (buttName[i]);
        }

        // Rang buoc cac nut nhan cach nhau 2 pixel
```

```
constraints.insets = new Insets(2,2,2,2);

// Qui dinh cac nut nhan se thay doi kich thuoc
// theo ca 2 chieu
constraints.fill = GridBagConstraints.BOTH;

// Rang buoc cho nut nhan thu 1
constraints.gridx = 1;
constraints.gridy = 1;
constraints.gridheight = 2;
constraints.gridwidth = 1;
layout.setConstraints(buttons[0], constraints);

// Rang buoc cho nut nhan thu 2
constraints.gridx = 2;
constraints.gridy = 1;
constraints.gridheight = 1;
constraints.gridwidth = 2;
layout.setConstraints(buttons[1], constraints);

// Rang buoc cho nut nhan thu 3
constraints.gridx = 2;
constraints.gridy = 2;
constraints.gridheight = 1;
constraints.gridwidth = 1;
layout.setConstraints(buttons[2], constraints);

// Rang buoc cho nut nhan thu 4
constraints.gridx = 1;
constraints.gridy = 3;
constraints.gridheight = 1;
constraints.gridwidth = 2;
layout.setConstraints(buttons[3], constraints);

// Rang buoc cho nut nhan thu 5
```

```
constraints.gridx = 3;
constraints.gridy = 2;
constraints.gridheight = 2;
constraints.gridwidth = 1;
layout.setConstraints(buttons[4], constraints);

// Rang buoc cho nut nhan thu 6
constraints.gridx = 4;
constraints.gridy = 1;
constraints.gridheight = 3;
constraints.gridwidth = 1;
layout.setConstraints(buttons[5], constraints);

// Tu nut thu 7 tro di khong can rang buoc
// thay vi doi kich thuoc
constraints.fill = GridBagConstraints.NONE;

// Rang buoc cho nut nhan thu 7
constraints.gridx = 1;
constraints.gridy = 4;
constraints.gridheight = 1;
constraints.gridwidth = 1;
constraints.weightx = 1.0;
layout.setConstraints(buttons[6], constraints);

// Rang buoc cho nut nhan thu 8
constraints.gridx = 2;
constraints.gridy = 5;
constraints.gridheight = 1;
constraints.gridwidth = 1;
constraints.weightx = 2.0;
layout.setConstraints(buttons[7], constraints);

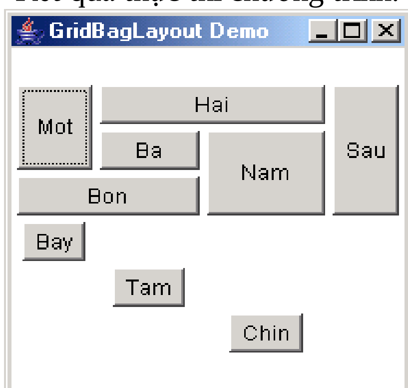
// Rang buoc cho nut nhan thu 9
constraints.gridx = 3;
```

```
        constraints.gridy = 6;
        constraints.gridheight = 1;
        constraints.gridwidth = 1;
        constraints.weightx = 3.0;
        layout.setConstraints(buttons[8], constraints);

        // Dua cac nut nhan khung chua chuong trinh
        for (int i=0;i<9;i++)
            f.add(buttons[i]);

        f.pack();
        f.setVisible(true);
    }
}
```

Kết quả thực thi chương trình:



4.4.4.5 Null Layout

Một khung chứa được trình bày theo kiểu Null Layout có nghĩa là người lập trình phải tự làm tất cả từ việc qui định kích thước của khung chứa, cũng như kích thước và vị trí của từng đối tượng component trong khung chứa.

Để thiết lập cách trình bày là Null Layout cho một container ta chỉ việc gọi phương thức setLayout(**null**) với tham số là **null.**

Một số phương thức của lớp trừu tượng Component dùng để định vị và qui định kích thước của component khi đưa chúng vào khung chứa trình bày theo kiểu kiểu tự do:

- Public void setLocation(Point p)
- Public void setSize(Dimension p)
- Public void setBounds(Rectangle r)

Ví dụ:

- MyButton.setSize(new Dimension(20, 10));
- MyButton.setLocation(new Point(10, 10));
- MyButton.setBounds(10, 10, 20, 10);

```
import java.awt.*;
class NullLayoutDemo
{
        public static void main(String args[])
        {
                Frame fr = new Frame("NullLayout Demo");
                fr.setLayout(null);
                Button buttOk = new Button("OK");
                buttOk.setBounds(100, 150, 50, 30);
                Button buttCancel = new Button("Cancel");
                buttCancel.setBounds(200, 150, 50, 30);
                Checkbox checkBut = new Checkbox("Check
                box", true);
                checkBut.setBounds(100, 50, 100, 20);

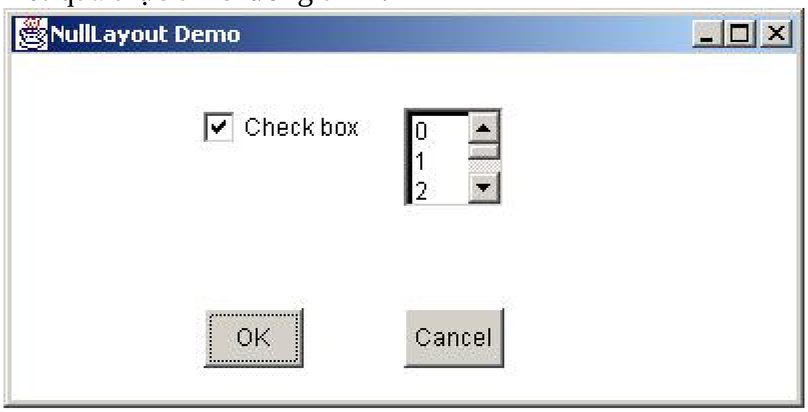
                List li = new List();
                for (int i=0; i<5; i++)
                {
                    li.add(Integer.toString(i));
                }
                li.setBounds(200, 50, 50, 50);

                fr.add(buttOk);
                fr.add(buttCancel);
```

```
            fr.add(checkBut);
            fr.add(li);

            fr.setBounds(10, 10, 400, 200);
            fr.setVisible(true);
        }
}
```

Kết quả thực thi chương trình:



4.4.5.Các đối tượng khung chứa Container

Như chúng ta đã biết container là đối tượng khung chứa có khả năng quản lý và chứa các đối tượng (components) khác trong nó.

Các components chỉ có thể sử dụng được khi đưa nó vào 1 đối tượng khung chứa là container.

Mỗi container thường gắn với một LayoutManager (FlowLayout, BorderLayout, GridLayout, GridBagLayout, Null Layout) qui định cách trình bày và bố trí các components trong một container.

Các loại container trong java: Frame, Panel, Dialog, ScrollPanes.

4.4.5.1 Khung chứa Frame

```
java.lang.Object
  +--java.awt.Component
      +--java.awt.Container
          +--java.awt.Window
                +--java.awt.Frame
```

Khung chứa Frame là một cửa số window hẳn hoi ở mức trên cùng bao gồm một tiêu đều và một đường biên (border) như các ứng dụng windows thông thường khác. Khung chứa Frame thường được sử dụng để tạo ra cửa sổ chính của các ứng dụng.

Khung chứa Panel có bộ quản lý trình bày (LayoutManager) mặc định là FlowLayout.

4.4.5.2 Khung chứa Panel

```
java.lang.Object
  +--java.awt.Component
      +--java.awt.Container
            +--java.awt.Panel
```

Khung chứa Panel có bộ quản lý trình bày (LayoutManager) mặc định là FlowLayout.

Đối với khung chứa Panel thì các Panel có thể lồng vào nhau, vì vậy khung chứa Panel thường được dùng để bố trí các nhóm components bên trong một khung chứa khác.

Ví dụ:

```
import java.awt.*;
public class PanelDemo extends Frame
{
    private Button next, prev, first;
    private List li;
    public PanelDemo(String sTitle)
    {
        super(sTitle);
        next = new Button("Next >>");
        prev = new Button("<< Prev");
```

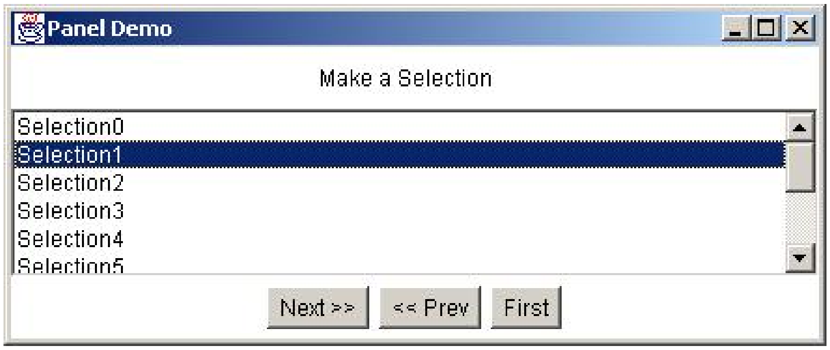
```
        first = new Button("First");

        Panel southPanel = new Panel();
        southPanel.add(next);
        southPanel.add(prev);
        southPanel.add(first);
        // BorderLayout.SOUTH: vùng dưới
        this.add(southPanel, BorderLayout.SOUTH);

        Panel northPanel = new Panel();
        northPanel.add(new Label("Make a Selection"));
        // BorderLayout.NORTH: vùng trên
        this.add(northPanel, BorderLayout.NORTH);

        li = new List();
        for(int i=0;i<10;i++)
        {
            li.add("Selection" + i);
        }
        this.add(li, BorderLayout.CENTER);
    }
    public static void main(String arg[])
    {
        Container f = new PanelDemo("Panel Demo");
        f.setSize(300, 200);
        f.setVisible(true);
    }
}
```

Kết quả thực thi chương trình:



4.4.5.2 Khung chứa Dialog

java.lang.Object
  +--java.awt.Component
    +--java.awt.Container
      +--java.awt.Window
        +--java.awt.Dialog

Dialog là một lớp khung chứa tựa Frame và còn được gọi là popup window. Có hai loại dialog phổ biến:

Modal Dialog: sẽ khóa tất cả các cửa số khác của ứng dụng khi dialog dạng này còn hiển thị.

Non-Modal Dialog: vẫn có thể đến các cửa số khác của ứng dụng khi dialog dạng này hiển thị.

Một cửa sổ dạng Dialog luôn luôn phải gắn với một cửa sổ ứng dụng (Frame).

Để tạo một đối tượng khung chứa Dialog ta có thể dùng một trong các constructor của nó:

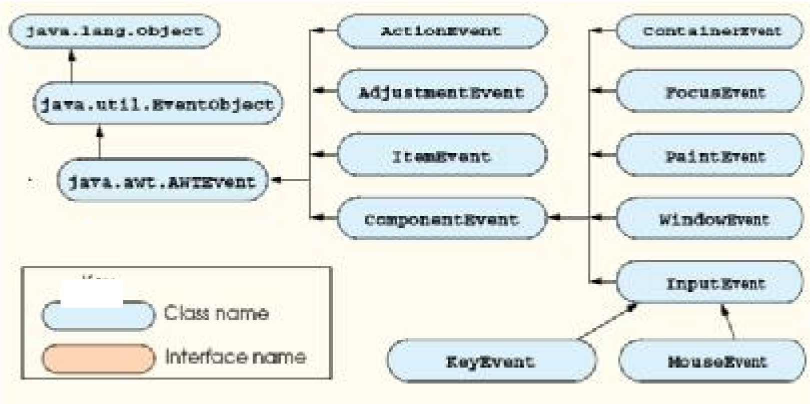
public Dialog (Frame parentWindow, boolean isModal)
public Dialog (Frame parentWindow, String title, boolean isModal)
parentWindow: cửa sổ cha
title: tiêu đề của Dialog
isModal: true -> là Dialog dạng modal
isModal: false -> là Dialog không phải dạng modal (hay non-modal)

4.5.Xử lý biến cố/sự kiện

4.5.1.Mô hình xử lý sự kiện (Event-Handling Model)

Ở trên chúng ta chỉ đề cập đến vấn đề thiết kế giao diện chương trình ứng dụng mà chưa đề cập đến vấn đề xử lý sự kiện. Những sự kiện được phát sinh khi người dùng tương tác với giao diện chương trình (GUI). Những tương tác thường gặp như: di chuyển, nhấn chuột, nhấn một nút nhấn, chọn một MenuItem trong hệ thống thực đơn, nhập dữ liệu trong một ô văn bản, đóng cửa sổ ứng dụng, … Khi có một tương tác xảy ra thì một sự kiện được gởi đến chương trình. Thông tin về sự kiện thường được lưu trữ trong một đối tượng dẫn xuất từ lớp AWTEvent. Những kiểu sự kiện trong gói **java.awt.event** có thể dùng cho cả những component AWT và JFC. Đối với thư viện JFC thì có thêm những kiểu sự kiện mới trong gói **java.swing.event**.

Những lớp sự kiện của gói java.awt.event



Có 3 yếu tố quan trọng trong mô hình xử lý sự kiện:

- Nguồn phát sinh sự kiện (event source)
- Sự kiện (event object)
- Bộ lắng nghe sự kiện (event listener)

Nguồn phát sinh sự kiện: là thành phần của giao diện mà người dùng tác động.

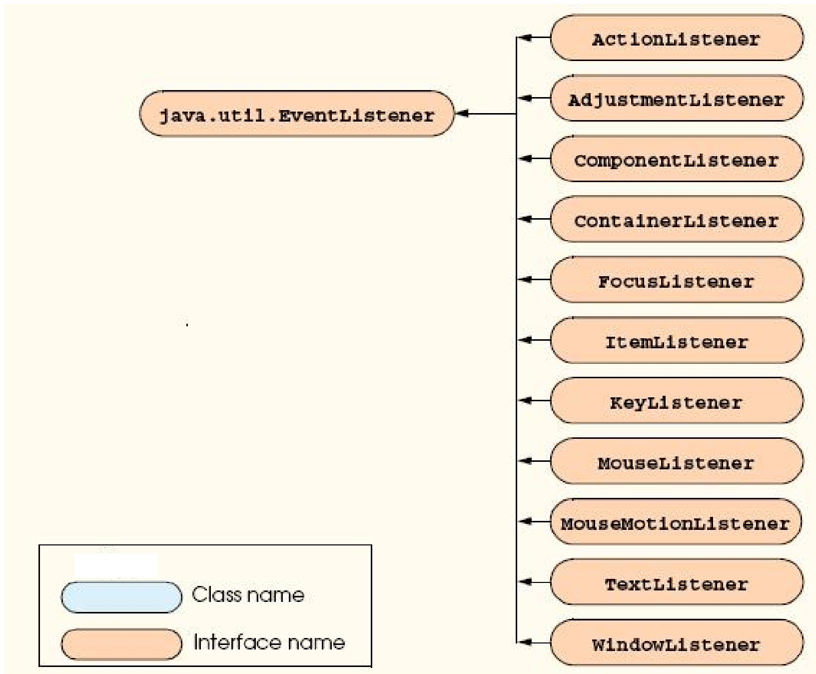
Sự kiện: Tóm tắt thông tin về xử kiện xảy ra, bao gồm tham chiếu đến nguồn gốc phát sinh sự kiện và thông tin sự kiện sẽ gởi đến cho bộ lắng nghe xử lý.

Bộ lắng nghe: Một bộ lắng nghe là một đối tượng của một lớp hiện thực một hay nhiều interface của gói **java.awt.event** hay java.swing.event (đối với những component trong thư viện JFC). Khi được thông báo, bộ lắng nghe nhận sự kiện và xử lý. Nguồn phát sinh sự kiện phải cung cấp những phương thức để đăng ký hoặc hủy bỏ một bộ lắng nghe. Nguồn phát sinh sự kiện luôn phải gắn với một bộ lắng nghe, và nó sẽ thông báo với bộ lắng nghe đó khi có sự kiện phát sinh đó.

Như vậy người lập trình cần làm hai việc:

- Tạo và đăng ký một bộ lắng nghe cho một component trên GUI.
- Cài đặt các phương thức quản lý và xử lý sự kiện

Những interfaces lắng nghe của gói java.awt.event



Một đối tượng Event-Listener lắng nghe những sự kiện khác nhau phát sinh từ các components của giao diện chương trình. Với mỗi sự kiện khác nhau phát sinh thì phương thức tương ứng trong những Event-Listener sẽ được gọi thực hiện.

Mỗi interface Event-Listener gồm một hay nhiều các phương thức mà chúng cần cài đặt trong các lớp hiện thực (implements) interface đó. Những phương thức trong các interface là trừu tượng vì vậy lớp (bộ lắng nghe) nào hiện thực các interface thì

phải cài đặt tất cả những phương thức đó. Nếu không thì các bộ lắng nghe sẽ trở thành các lớp trừu tượng.

4.5.2.Xử lý sự kiện chuột

Java cung cấp hai intefaces lắng nghe (bộ lắng nghe sự kiện chuột) là MouseListener và MouseMotionListener để quản lý và xử lý các sự kiện liên quan đến thiết bị chuột. Những sự kiện chuột có thể “bẫy” cho bất kỳ component nào trên GUI mà dẫn xuất từ java.awt.component.

Các phương thức của interface MouseListener:

- *public void* ***mousePressed(MouseEvent event):*** được gọi khi một nút chuột được nhấnvà con trỏ chuột ở trên component.
- *public void* ***mouseClicked(MouseEvent event):*** được gọi khi một nút chuột được nhấn và nhả trên component mà không di chuyển chuột.
- *public void* ***mouseReleased(MouseEvent event):*** được gọi khi một nút chuột nhả sa khi kéo rê.
- *public void* ***mouseEntered(MouseEvent event):*** được gọi khi con trỏ chuột vào trong đường biên của một component.
- *public void* ***mouseExited(MouseEvent event)***: được gọi khi con trỏ chuột ra khỏi đường biên của một component.

Các phương thức của interface MouseMotionListener:

- *public void* ***mouseDragged(MouseEvent even):*** phương thức này được gọi khi người dùng nhấn một nút chuột và kéo trên một component.
- *public void* ***mouseMoved(MouseEvent event)***: phương thức này được gọi khi di chuyển chuột trên component.

Mỗi phương thức xử lý sự kiện chuột có một tham số MouseEvent chứa thông tin về sự kiện chuột phát sinh chẳng hạn như: tọa độ x, y nơi sự kiện chuột xảy ra. Những phương

thức tương ứng trong các interfaces sẽ tự động được gọi khi chuột tương tác với một component.

Để biết được người dùng đã nhấn nút chuột nào, chúng ta dùng những phương thức, những hằng số của lớp InputEvent (là lớp cha của lớp MouseEvent).

Ví dụ: Chương trình tên MouseTracker bên dưới minh họa việc dùng những phương thức của các interfaces MouseListener và MouseMotionListener để "bẫy" và xử lý các sự kiện chuột tương ứng.

```
import java.awt.*;
import java.awt.event.*;
public class MouseTracker extends Frame
    implements MouseListener, MouseMotionListener
{
    private Label statusBar;
    // set up GUI and register mouse event handlers
    public MouseTracker()
    {   super( "Demonstrating Mouse Events" );
        statusBar = new Label();
        this.add( statusBar, BorderLayout.SOUTH );
        // application listens to its own mouse events
        addMouseListener( this );
        addMouseMotionListener( this );
        setSize( 275, 100 );
        setVisible( true );
    }

    // MouseListener event handlers
    // handle event when mouse released immediately
    // after press
    public void mouseClicked( MouseEvent event )
    {
        statusBar.setText( "Clicked at [" + event.getX() +
```

```
                ", " + event.getY() + "]" );
}

// handle event when mouse pressed
public void mousePressed( MouseEvent event )
{
     statusBar.setText( "Pressed at [" + event.getX() +
                ", " + event.getY() + "]" );
}

// handle event when mouse released after dragging
public void mouseReleased( MouseEvent event )
{
     statusBar.setText( "Released at [" + event.getX() +
                ", " + event.getY() + "]" );
}

// handle event when mouse enters area
public void mouseEntered( MouseEvent event )
{
     statusBar.setText( "Mouse in window" );
}

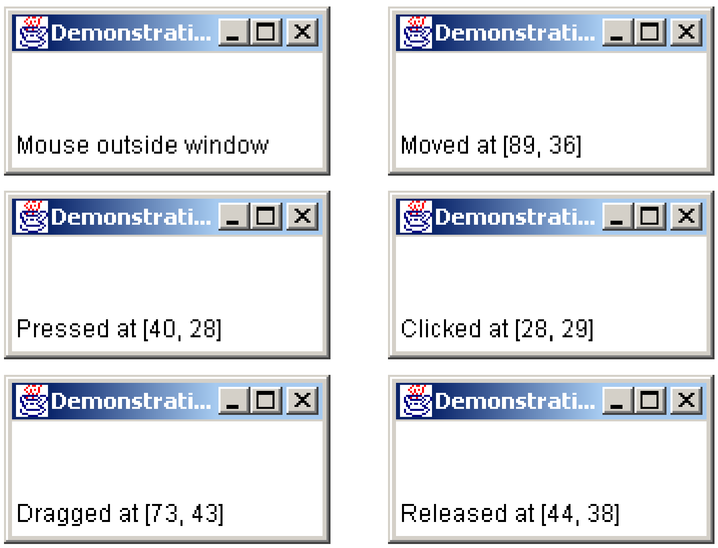
// handle event when mouse exits area
public void mouseExited( MouseEvent event )
{    statusBar.setText( "Mouse outside window" );
}

// MouseMotionListener event handlers
// handle event when user drags mouse with button pressed
public void mouseDragged( MouseEvent event )
{
     statusBar.setText( "Dragged at [" + event.getX() +
                ", " + event.getY() + "]" );
}
```

```
    // handle event when user moves mouse
    public void mouseMoved( MouseEvent event )
    {
        statusBar.setText( "Moved at [" + event.getX() +
                ", " + event.getY() + "]" );
    }

    // execute application
    public static void main( String args[] )
    {
        MouseTracker application = new MouseTracker();
    }
} // end class MouseTracker
```

Kết quả thực thi chương trình:



4.5.3.Xử lý sự kiện bàn phím

Để xử lý sự kiện bàn phím java hỗ trợ một bộ lắng nghe sự kiện đó là interface *KeyListener*. Một sự kiện bàn phím được

phát sinh khi người dùng nhấn và nhả một phím trên bàn phím. Một lớp hiện thực *KeyListener* phải cài đặt các phương thức *keyPressed*, *keyReleased* và *keyTyped*. Mỗi phương thức này có một tham số là một đối tượng kiểu *KeyEvent*. *KeyEvent* là lớp con của lớp *InputEvent*.

Các phương thức của interface *KeyListener*

- Phương thức *keyPressed* được gọi khi một phím bất kỳ được nhấn.
- Phương thức *keyTyped* được gọi thực hiện khi người dùng nhấn một phím không phải "phím hành động" (như phím mũi tên, phím Home, End, Page Up, Page Down, các phím chức năng như: Num Lock, Print Screen, Scroll Lock, Caps Lock, Pause).
- Phương thức *keyReleased* được gọi thực hiện khi nhả phím nhấn sau khi sự kiện *keyPressed* hoặc *keyTyped*.

Ví dụ: minh họa việc xử lý sự kiện chuột thông qua các phương thức của interface *KeyListener*. Lớp *KeyDemo* bên dưới hiện thực interface *KeyListener*, vì vậy tất cả 3 phương thức trong *KeyListener* phải được cài đặt trong chương trình.

```
// KeyDemo.java
// Demonstrating keystroke events.
// Java core packages
import java.awt.*;
import java.awt.event.*;
public class KeyDemo extends Frame implements KeyListener
{
    private String line1 = "", line2 = "";
    private String line3 = "";
    private TextArea textArea;

    // set up GUI
    public KeyDemo()
```

```
{
    super( "Demonstrating Keystroke Events" );

    // set up TextArea
    textArea = new TextArea( 10, 15 );
    textArea.setText( "Press any key on the keyboard..." );
    textArea.setEnabled( false );
    this.add( textArea );

    // allow frame to process Key events
    addKeyListener( this );

    setSize( 350, 100 );
    setVisible( true );
}

// handle press of any key
public void keyPressed( KeyEvent event )
{
    line1 = "Key pressed: " +
    event.getKeyText( event.getKeyCode() );
    setLines2and3( event );
}

// handle release of any key
public void keyReleased( KeyEvent event )
{
    line1 = "Key released: " +
    event.getKeyText( event.getKeyCode() );
    setLines2and3( event );
}

// handle press of an action key
public void keyTyped( KeyEvent event )
{
```

```
        line1 = "Key typed: " + event.getKeyChar();
        setLines2and3( event );
    }

    // set second and third lines of output
    private void setLines2and3( KeyEvent event )
    {
        line2 = "This key is " + ( event.isActionKey() ? "" : "not
        " ) + "an action key";

        String temp = event.getKeyModifiersText(
        event.getModifiers() );

        line3 = "Modifier keys pressed: " + ( temp.equals( "" ) ?
        "none" : temp );

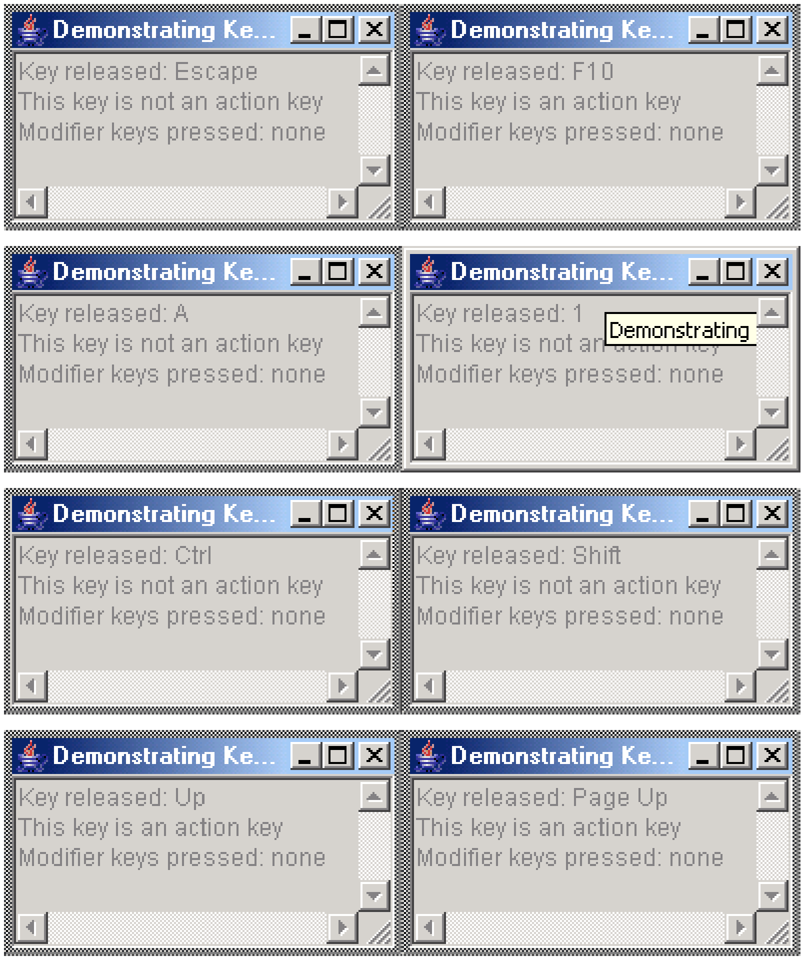
        textArea.setText(line1+"\n"+line2+"\n"+ line3+"\n" );
    }

    // execute application
    public static void main( String args[] )
    {
        KeyDemo application = new KeyDemo();
    }

} // end class KeyDemo
```

Kết quả thực thi chương trình:





4.6.Một số ví dụ minh họa

Ví dụ 1: Tạo bộ lắng nghe biến cố cho đối tượng khung chứa Frame, và xử lý biến cố đóng cửa sổ.

```
import java.awt.*;
import java.awt.event.*;
public class WindowClosingDemo
{
    public static void main(String args[])
```

```
    {
        Frame f = new Frame ("WindowClosing Demo");
        WindowCloser closer = new WindowCloser();
        f.addWindowListener(closer);
        f.setBounds(10, 10, 300, 200);
        f.setVisible(true);
    }
}

import java.awt.event.*;
class WindowCloser implements WindowListener
{
    public void windowClosing(WindowEvent e)
    {
        System.out.println("windowClosing..");
        System.exit(0);
    }
    public void windowActivated(WindowEvent e)
    {
        System.out.println("windowActivated...");
    }
    public void windowClosed(WindowEvent e)
    {
        System.out.println("windowClosed...");
    }
    public void windowDeactivated(WindowEvent e)
    {
        System.out.println("windowDeactivated...");
    }
    public void windowDeiconified(WindowEvent e)
    {
        System.out.println("windowDeiconified...");
    }
    public void windowIconified(WindowEvent e)
    {
```

```
        System.out.println("windowIconified...");
    }
    public void windowOpened(WindowEvent e)
    {   System.out.println("windowOpened...");
    }
}
```

Có thể dùng lớp trừu tượng WindowAdapter để tạo ra bộ lắng nghe.

public abstract class **WindowAdapter** extends [Object]
implements [WindowListener]

(**WindowAdapter** hiện thực interface WindowListener nên lớp ảo này cũng có 7 phương thức giống như giao diện WindowListener)

```
import java.awt.event.*;
class WindowCloser extends WindowAdapter
{   public void windowClosing(WindowEvent e)
    {   System.out.println("windowClosing..");
        System.exit(0);
    }
}
```

Ví dụ 2: CheckboxGroup Demo

```
import java.awt.*;
public class CheckboxGroupDemo extends Frame
{
    private Checkbox red, green, blue;
    private CheckboxGroup checkGroup;
    public CheckboxGroupDemo(String title)
    {   super(title);
        checkGroup = new CheckboxGroup();
        red = new Checkbox("Red", checkGroup, false);
        green = new Checkbox("Green", checkGroup, false);
        blue = new Checkbox("Blue", checkGroup, false);
```

```
        //add the checkboxes to the frame
        Panel north = new Panel();
        north.add(red);
        north.add(green);
        north.add(blue);
        this.add(north, BorderLayout.NORTH);

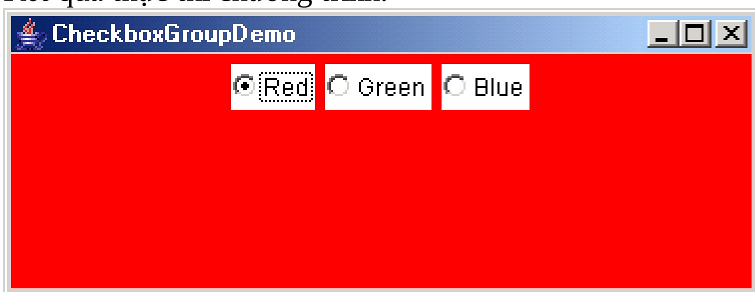
        //register the event listener
        SetColor listener = new SetColor(this);
        red.addItemListener(listener);
        green.addItemListener(listener);
        blue.addItemListener(listener);
    }

    public static void main(String [] args)
    {
        Frame f = new
        CheckboxGroupDemo("CheckboxGroupDemo");
        f.setSize(300,300);
        f.setVisible(true);
    }
} // end of class

import java.awt.*;
import java.awt.event.*;
public class SetColor implements ItemListener
{
    private Frame pallette;
    private Color c;
    public SetColor(Frame c)
    {
        pallette = c;
    }
```

```
    public void itemStateChanged(ItemEvent e)
    {
        String item = (String) e.getItem();
        int state = e.getStateChange();
        if (item.equalsIgnoreCase("red"))
                c = new Color(255, 0, 0);
        if (item.equalsIgnoreCase("green"))
                c = new Color(0, 255, 0);
        if (item.equalsIgnoreCase("blue"))
                c = new Color(0, 0, 255);
        pallette.setBackground(c);
    }
} // end of class
```

Kết quả thực thi chương trình:



Ví dụ 3: TextComponent

```
import java.awt.*;
class TextComponentDemo extends Frame
{
    private TextField textField;
    private TextArea textArea;
    private Button enter, clear;

    public TextComponentDemo (String title)
    {
        super(title);
```

```
        textArea = new TextArea("", 0, 0,
        TextArea.SCROLLBARS_VERTICAL_ONLY);
        textArea.setEditable(false);

        textField = new TextField();
        enter = new Button("Enter");
        clear = new Button("Clear");

        //layout the GUI
        this.add(textArea, BorderLayout.CENTER);

        Panel southEast = new Panel(new BorderLayout());
        southEast.add(enter, BorderLayout.EAST);
        southEast.add(clear, BorderLayout.WEST);

        Panel south = new Panel(new BorderLayout());
        south.add(textField, BorderLayout.CENTER);
        south.add(southEast, BorderLayout.EAST);

        this.add(south, BorderLayout.SOUTH);

        //setup the event handling
        CreateList listener = new CreateList(textField,
        textArea);
        enter.addActionListener(listener);
        clear.addActionListener(listener);
        textField.addActionListener(listener);
    }

    public TextField getTextField()
    {
        return textField;
    }

    public static void main(String [] args)
```

```
    {
        TextComponentDemo f = new TextComponentDemo
        ("TextComponentDemo ");
        f.setSize(300,200);
        f.setVisible(true);
        f.getTextField().requestFocus();
    }
}

import java.awt.*;
import java.awt.event.*;
public class CreateList implements ActionListener
{
    private int counter = 0;
    private TextField source;
    private TextArea destination;

    public CreateList(TextField s, TextArea d)
    {   source = s;
        destination = d;
    }

    public void actionPerformed(ActionEvent e)
    {
        String action = e.getActionCommand();
        if (action.equalsIgnoreCase("Enter"))
        {
            String text = source.getText();
            counter++;
            destination.append(counter + "." + text + "\n");
            source.setText("");
        }
        else
            if (action.equalsIgnoreCase("Clear"))
            {
```

```
                destination.setText("");
                counter = 0;
            }
        }
}
```

Kết quả thực thi chương trình:



Ví dụ 4: ListDemo

```
import java.awt.*;
public class ListDemo extends Frame
{   private List li;
    private Label selected;
    public ListDemo(String title)
    {
        super(title);
        li = new List();
        li.add("Monday");
        li.add("Tuesday");
        li.add("Wednesday");
        li.add("Thursday");
        li.add("Friday");
        li.add("Saturday");
        li.add("Sunday");
```

```
        selected = new Label("Double click a day:",
        Label.CENTER);
        this.setLayout(new BorderLayout());

        this.add(selected , BorderLayout.NORTH);
        this.add(li, BorderLayout.CENTER);

        // Tao listener cho List
        ShowSelectionListener listener = new
        ShowSelectionListener(selected);
        li.addActionListener(listener);
    }

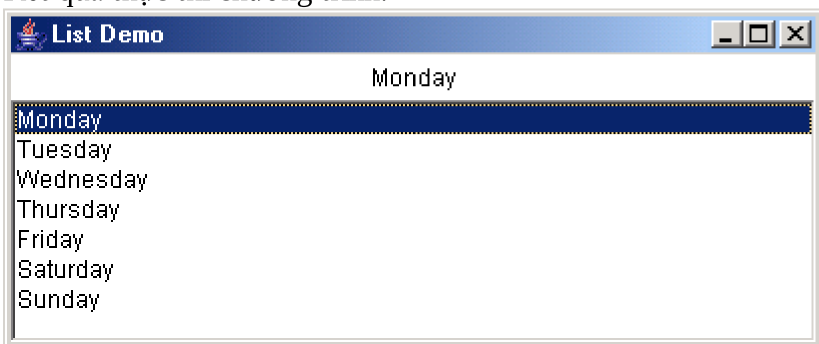
    public static void main(String args[])
    {   ListDemo f = new ListDemo("List Demo");
        f.setBounds(10, 10, 300, 200);
        f.setVisible(true);
    }
}

import java.awt.*;
import java.awt.event.*;
class ShowSelectionListener implements ActionListener
{   private Label lab;
    public ShowSelectionListener(Label label_sel)
    {
        lab = label_sel;
    }

    public void actionPerformed(ActionEvent e)
    {   // Tra ve Object ma Event da xuat hien
        // getSource la phuong thuc ke thua tu
        // java.util.EventObject
        Object source = e.getSource();
```

```
            // Nguon goc phat sinh bien co khong phai la List
            if (!(source instanceof List))
            {    return;
            }
            else
            {
                List li = (List) source;
                String selected = li.getSelectedItem();
                lab.setText(selected);
            }
        }
}
```

Kết quả thực thi chương trình:



Ví dụ 5: Xây dựng 1 lớp khung chứa Dialog dùng để hiển thị message giống như hàm MessageBox trên Windows.

```
import java.awt.*;
import java.awt.event.*;
class DialogDemo
{
    public static void main(String[] args)
    {
        createMenu();
    }
```

```
private static void createMenu()
{
    // Tao Frame ung dung
    final Frame fr = new Frame();
    fr.setLayout(new BorderLayout());

    // Tao cac menu bar
    MenuBar menubar = new MenuBar();
    Menu mTest = new Menu("Test");
    MenuItem testDlg = new MenuItem("Test Dialog");
    testDlg.addActionListener(
        new ActionListener()
        {
            public void actionPerformed(ActionEvent e)
            {
                    MessageBox msgBox = new
                    MessageBox(fr, "Here it is", "T/bao
                    Dialog");
                    msgBox.show();
            }
        }
    );

    mTest.add(testDlg);
    menubar.add(mTest);
    fr.setMenuBar(menubar);
    fr.setBounds(100, 100, 300, 200);
    fr.setVisible(true);

    fr.addWindowListener(
            new WindowAdapter()
            {
                public void windowClosing(WindowEvent e)
                {
                    System.exit(0);
```

```
                    }
                }
            );

        }// end of createmenu()
} // end of class

import java.awt.*;
import java.awt.event.*;
public class MessageBox
{
    Dialog msgBox;
    /* ------------------------------------------------------------
    // Contructor cua lop MessageBox
    // parentWindow: cua so cha
    // title: Tieu de cua Dialog
    // msg: chuoi thong bao
    --------------------------------------------------------------*/
    public MessageBox(Frame parentWindow, String msg,
    String title)
    {
        if (parentWindow == null)
        {
            Frame emptyWin = new Frame();
            // Tao Modal Dialog (tham so thu 3:true)
            msgBox = new Dialog(emptyWin, title, true);
        }
        else
        {
            msgBox = new Dialog(parentWindow, title, true);
        }

        // Doi tuong nhan dung de trinh bay cau thong bao
        Label Message = new Label(msg);
        // Thiet lap che do trinh bay layout cho cac doi tuong.
```

```java
            msgBox.setLayout(new FlowLayout());
            // Dua nhan thong bao Label vao khung chua Dialog
            msgBox.add(Message);
            // Dua nut nhan OK vao trong khung chua Dialog
            Button okButton = new Button("OK");
            msgBox.add(okButton);
            // Khai bao kich thuoc cua cua so thong bao
            msgBox.setSize(200, 100);

            // Xu ly tinh huong khi nguoi dung nhan nut OK
            okButton.addActionListener(
                new ActionListener()
                {
                    public void actionPerformed(ActionEvent evt)
                    {
                            msgBox.setVisible(false);
                    }
                }
            );
        }

        public void show()
        {
            msgBox.show();
        }

} // end of class MessageBox
```

Kết quả thực thi chương trình:



Chương 5: LUỒNG VÀ TẬP TIN

(STREAMS & FILES)

5.1.Mở đầu

Việc lưu trữ dữ liệu trong các biến chương trình, các mảng có tính chất tạm thời và dữ liệu sẽ mất đi khi biến ra khỏi tầm ảnh hưởng của nó hoặc khi chương trình kết thúc. Files giúp cho các chương trình có thể lưu trữ một lượng lớn dữ liệu, cũng như có thể lưu trữ dữ liệu trong một thời gian dài ngay cả khi chương trình kết thúc. Trong chương này chúng ta sẽ tìm hiểu làm thế nào các chương trình java có thể tạo, đọc, ghi và xử lý các files tuần tự và các file truy cập ngẫu nhiên thông qua một số ví dụ minh họa.

Xử lý files là một vấn đề hết sức cơ bản, quan trọng mà bất kỳ một ngôn ngữ lập trình nào cũng phải hỗ trợ những thư viện, hàm để xử lý một số thao tác cơ bản nhất đối với kiểu dữ liệu file.

Xử lý files là một phần của công việc xử lý các luồng, giúp cho một chương trình có thể đọc, ghi dữ liệu trong bộ nhớ, trên files và trao đổ dữ liệu thông qua các kết nối trên mạng.

Chương này sẽ cung cấp cho chúng ta những kiến thức cơ bản về luồng (streams) và files:

- Thư viện các lớp về luồng trong java: luồng byte, luồng ký tự.
- Xuất nhập Console dùng luồng byte, luồng ký tự.
- Xuất nhập files dùng luồng ký tự và luồng byte.
- Vấn đề xử lý files truy cập ngẫu nhiên dùng lớp RandomAccessFile.
- Xử lý file và thư mục dùng lớp File.

5.2.Luồng (Streams)

5.2.1.Khái niệm luồng

Tất cả những hoạt động nhập/xuất dữ liệu (nhập dữ liệu từ bàn phím, lấy dữ liệu từ mạng về, ghi dữ liệu ra đĩa, xuất dữ liệu ra màn hình, máy in, …) đều được quy về một khái niệm gọi là luồng (stream). Luồng là nơi có thể "sản xuất" và "tiêu thụ" thông tin. Luồng thường được hệ thống xuất nhập trong java gắn kết với một thiết bị vật lý. Tất cả các luồng đều có chung một nguyên tắc hoạt động ngay cả khi chúng được gắn kết với các thiết bị vật lý khác nhau. Vì vậy cùng một lớp, phương thức xuất nhập có thể dùng chung cho các thiết bị vật lý khác nhau. Chẳng hạn cùng một phương thức có thể dùng để ghi dữ liệu ra console, đồng thời cũng có thể dùng để ghi dữ liệu xuống một file trên đĩa. Java hiện thực luồng bằng tập hợp các lớp phân cấp trong gói *java.io*.

Java định nghĩa hai kiểu luồng: byte và ký tự (phiên bản gốc chỉ định nghĩa kiểu luồng byte, và sau đó luồng ký tự được thêm vào trong các phiên bản về sau).

Luồng byte (hay luồng dựa trên byte) hỗ trợ việc xuất nhập dữ liệu trên byte, thường được dùng khi đọc ghi dữ liệu nhị phân.

Luồng ký tự được thiết kế hỗ trợ việc xuất nhập dữ liệu kiểu ký tự (Unicode). Trong một vài trường hợp luồng ký tự sử dụng hiệu quả hơn luồng byte, nhưng ở mức hệ thống thì tất cả những xuất nhập đều phải qui về byte. Luồng ký tự hỗ trợ hiệu quả chỉ đối với việc quản lý, xử lý các ký tự.

5.2.2.Luồng byte (Byte Streams)

Các luồng byte được định nghĩa dùng hai lớp phân cấp. Mức trên cùng là hai lớp trừu tượng *InputStream* và *OutputStream*. *InputStream* định nghĩa những đặc điểm chung cho những luồng nhập byte. *OutputStream* mô tả cách xử lý của các luồng xuất byte.

Các lớp con dẫn xuất từ hai lớp *InputStream* và *OutputStream* sẽ hỗ trợ chi tiết tương ứng với việc đọc ghi dữ liệu trên những thiết bị khác nhau. Đừng choáng ngợp với hàng loạt rất nhiều các lớp khác nhau. Đừng quá lo lắng, mỗi khi bạn nắm vững, sử dụng thành thạo một luồng byte nào đó thì bạn dễ dàng làm việc với những luồng còn lại.

Lớp luồng byte	Ý nghĩa
BufferedInputStream	Buffered input stream
BufferedOutputStream	Buffered output stream
ByteArrayInputStream	Input stream đọc dữ liệu từ một mảng byte
ByteArrayOutputStream	Output stream ghi dữ liệu đến một mảng byte
DataInputStream	Luồng nhập có những phương thức đọc những kiểu dữ liệu chuẩn trong java
DataOutputStream	Luồng xuất có những phương thức ghi những kiểu dữ liệu chuẩn trong java
FileInputStream	Luồng nhập cho phép đọc dữ liệu từ file
FileOutputStream	Luồng xuất cho phép ghi dữ liệu xuống file
FilterInputStream	Hiện thực lớp trừu tượng *InputStream*
FilterOutputStream	Hiện thực lớp trừu tượng *OutputStream*
InputStream	Lớp trừu tượng, là lớp cha của tất cả các lớp luồng nhập kiểu Byte
OutputStream	Lớp trừu tượng, là lớp cha của tất cả các lớp xuất nhập kiểu Byte
PipedInputStream	Luồng nhập byte kiểu ống (piped) thường phải được gắn với một luồng xuất kiểu ống.

PipedOutputStream	Luồng nhập byte kiểu ống (piped) thường phải được gắn với một luồng nhập kiểu ống để tạo nên một kết nối trao đổi dữ liệu kiểu ống.
PrintStream	Luồng xuất có chứa phương thức *print()* và *println()*
PushbackInputStream	Là một luồng nhập kiểu Byte mà hỗ trợ thao tác trả lại (push back) và phục hồi thao tác đọc một byte (unread)
RandomAccessFile	Hỗ trợ các thao tác đọc, ghi đối với file truy cập ngẫu nhiên.
SequenceInputStream	Là một luồng nhập được tạo nên bằng cách nối kết logic các luồng nhập khác.

5.2.3.Luồng ký tự (Character Streams)

Các luồng ký tự được định nghĩa dùng hai lớp phân cấp. Mức trên cùng là hai lớp trừu tượng *Reader* và *Writer*. Lớp *Reader* dùng cho việc nhập dữ liệu của luồng, lớp *Writer* dùng cho việc xuất dữ liệu cua luồng. Những lớp dẫn xuất từ *Reader* và *Writer* thao tác trên các luồng ký tự Unicode.

Lớp luồng ký tự	Ý nghĩa
BufferedReader	Luồng nhập ký tự đọc dữ liệu vào một vùng đệm.
BufferedWriter	Luồng xuất ký tự ghi dữ liệu tới một vùng đệm.
CharArrayReader	Luồng nhập đọc dữ liệu từ một mảng ký tự
CharArrayWriter	Luồng xuất ghi dữ liệu tời một mảng ký tự

FileReader	Luồng nhập ký tự đọc dữ liệu từ file
FileWriter	Luồng xuất ký tự ghi dữ liệu đến file
FilterReader	Lớp đọc dữ liệu trung gian (lớp trừu tượng)
FilterWriter	Lớp xuất trung gian trừu tượng
InputStreamReader	Luồng nhập chuyển bytes thành các ký tự
LineNumberReader	Luồng nhập đếm dòng
OutputStreamWriter	Luồng xuất chuyển những ký tự thành các bytes
PipedReader	Luồng đọc dữ liệu bằng cơ chế đường ống
PipedWriter	Luồng ghi dữ liệu bằng cơ chế đường ống
PrintWriter	Luồng ghi văn bản ra thiết bị xuất (chứa phương thức *print()* và *println()*)
PushbackReader	Luồng nhập cho phép đọc và khôi phục lại dữ liệu
Reader	Lớp nhập dữ liệu trừu tượng
StringReader	Luồng nhập đọc dữ liệu từ chuỗi
StringWriter	Luồng xuất ghi dữ liệu ra chuỗi
Writer	Lớp ghi dữ liệu trừu tượng

5.2.4.Những luồng được định nghĩa trước (The Predefined Streams)

Tất cả các chương trình viết bằng java luôn tự động import gói *java.lang*. Gói này có định nghĩa lớp *System*, bao gồm một số đặc điểm của môi trường run-time, nó có ba biến luồng được định nghĩa trước là ***in***, ***out*** và ***err***, các biến này là các fields được khai báo *static* trong lớp *System*.

- System.out: luồng xuất chuẩn, mặc định là console. System.out là một đối tượng kiểu *PrintStream*.
- System.in: luồng nhập chuẩn, mặc định là bàn phím. System.in là một đối tượng kiểu *InputStream*.
- System.err: luồng lỗi chuẩn, mặc định cũng là console. System.out cũng là một đối tượng kiểu *PrintStream* giống System.out.

5.3.Sử dụng luồng Byte

Như chúng ta đã biết hai lớp *InputStream* và *OutputStream* là hai siêu lớp (cha) đối với tất cả những lớp luồng xuất nhập kiểu byte. Những phương thức trong hai siêu lớp này ném ra các lỗi kiểu IOException. Những phương thức định nghĩa trong hai siêu lớp này là có thể dùng trong các lớp con của chúng. Vì vậy tập các phương thức đó là tập tối tiểu các chức năng nhập xuất mà những luồng nhập xuất kiểu byte có thể sử dụng.

Những phương thức định nghĩa trong lớp

InputStream và OutputStream

Phương thức	Ý nghĩa
InputStream	
int available( )	Trả về số lượng bytes có thể đọc được từ luồng nhập
void close( )	Đóng luồng nhập và giải phóng tài nguyên hệ thống gắn với luồng. Không thành công sẽ ném ra một lỗi IOException
void mark(int numBytes)	Đánh dấu ở vị trí hiện tại trong luồng nhập
boolean markSupported( )	Kiểm tra xem luồng nhập có hỗ trợ phương thức *mark()* và *reset()* không.

int read( )	Đọc byte tiếp theo từ luồng nhập
int read(byte buffer[ ])	Đọc buffer.length bytes và lưu vào trong vùng nhớ buffer. Kết quả trả về số bytes thật sự đọc được
int read(byte buffer[ ], int offset, int numBytes)	Đọc numBytes bytes bắt đầu từ địa chỉ offset và lưu vào trong vùng nhớ buffer. Kết quả trả về số bytes thật sự đọc được
void reset( )	Nhảy con trỏ đến vị trí được xác định bởi việc gọi hàm *mark()* lần sau cùng.
long skip(long numBytes)	Nhảy qua numBytes dữ liệu từ luồng nhập
OutputStream	
void close( )	Đóng luồng xuất và giải phóng tài nguyên hệ thống gắn với luồng. Không thành công sẽ ném ra một lỗi IOException
void flush( )	Ép dữ liệu từ bộ đệm phải ghi ngay xuống luồng (nếu có)
void write(int b)	Ghi byte dữ liệu chỉ định xuống luồng
void write(byte buffer[ ])	Ghi buffer.length bytes dữ liệu từ mảng chỉ định xuống luồng
void write(byte buffer[ ], int offset, int numBytes)	Ghi numBytes bytes dữ liệu từ vị trí offset của mảng chỉ định buffer xuống luồng

5.3.1.Đọc dữ liệu từ Console

Trước đây, khi Java mới ra đời để thực hiện việc nhập dữ liệu từ Console người ta chỉ dùng luồng nhập byte. Về sau thì

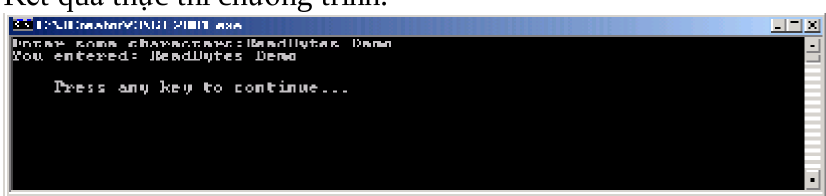
chúng ta có thể dùng cả luồng byte và luồng ký tự, nhưng trong một số trường hợp thực tế để đọc dữ liệu từ Console người ta thích dùng luồng kiểu ký tự hơn, vì lý do đơn giản và dễ bảo trì chương trình. Ở đây với mục đích minh họa chúng ta dùng luồng byte thực hiện việc nhập xuất Console.

Ví dụ: chương trình minh họa việc đọc một mảng bytes từ System.in

```
Import java.io.*;

class ReadBytes
{
        public static void main(String args[])
                throws IOException
        {
                byte data[] = new byte[100];
                System.out.print("Enter some characters.");
                System.in.read(data);
                System.out.print("You entered: ");
                for(int i=0; i < data.length; i++)
                        System.out.print((char) data[i]);
        }
}
```

Kết quả thực thi chương trình:



5.3.2.Xuất dữ liệu ra Console

Tương tự như nhập dữ liệu từ Console, với phiên bản đầu tiên của java để xuất dữ liệu ra Console tả chỉ có thể sử dụng

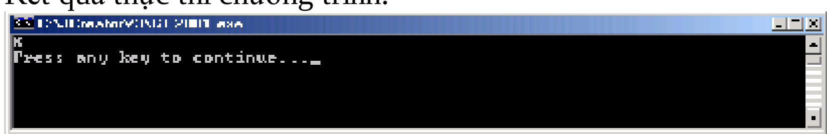
luồng byte. Kể từ phiên bản 1.1 (có thêm luồng ký tự), để xuất dữ liệu ra Console có thể sử dụng cả luồng ký tự và luồng byte. Tuy nhiên, cho đến nay để xuất dữ liệu ra Console thường người ta vẫn dùng luồng byte.

Chúng ta đã khá quen thuộc với phương thức print() và println(), dùng để xuất dữ liệu ra Console. Bên cạnh đ1o chúng ta cũng có thể dùng phương thức *write()*.

Ví dụ: minh họa sử dụng phương thức System.out.write() để xuất ký tự 'X' ra Console

```
import java.io.*;
class WriteDemo
{
    public static void main(String args[])
    {
        int b;
        b = 'X';
        System.out.write(b);
        System.out.write('\n');
    }
}
```

Kết quả thực thi chương trình:



5.3.3.Đọc và ghi file dùng luồng Byte

Tạo một luồng Byte gắn với file chỉ định dùng FileInputStream và FileOutputStream. Để mở một file, đơn giản chỉ cần tạo một đối tượng của những lớp này, tên file cần mở là thông số trong constructor. Khi file mở, việc đọc và ghi dữ liệu

trên file được thực hiện một cách bình thường thông qua các phương thức cung cấp trong luồng.

5.3.3.1 Đọc dữ liệu từ file

- Mở một file để đọc dữ liệu
  FileInputStream(String fileName) throws FileNotFoundException
  Nếu file không tồn tại: thì ném ra FileNotFoundException
- Đọc dữ liệu: dùng phương thức *read()*
  int read( ) throws IOException: đọc từng byte từ file và trả về giá trị của byte đọc được. Trả về -1 khi hết file, và ném ra IOException khi có lỗi đọc.
- Đóng file: dùng phương thức *close()*
  void close( ) throws IOException: sau khi làm việc xong cần đóng file để giải phóng tài nguyên hệ thống đã cấp phát cho file.

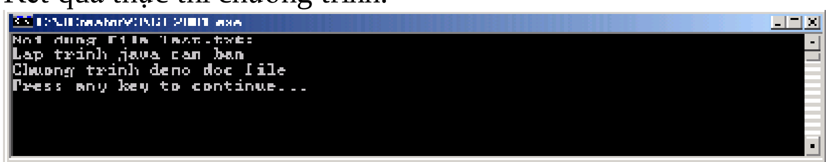
Ví dụ:

```
/*
Hiển thị nội dung của một file tên test.txt lưu tạiD:\test.txt
*/
import java.io.*;
class ShowFile
 {
    public static void main(String args[]) throws IOException
    {
        int i;
        FileInputStream fin;
        try
        {
            fin = new FileInputStream("D:\\test.txt");
        }
         catch(FileNotFoundException exc)
        {
                System.out.println("File Not Found");
```

```
            return;
        }
        catch(ArrayIndexOutOfBoundsException exc)
         {
            System.out.println("Usage: ShowFile File");
            return;
        }

        // read bytes until EOF is encountered
        do
         {
            i = fin.read();
            if(i != -1) System.out.print((char) i);
        } while(i != -1);
        fin.close();
    }
}
```

Kết quả thực thi chương trình:



5.3.3.2 Ghi dữ liệu xuống file

- Mở một file để ghi dữ liệu
  FileOutputStream(String fileName) throws FileNotFoundException
  Nếu file không tạo được: thì ném ra FileNotFoundException
- Ghi dữ liệu xuống: dùng phương thức *write()*
  void write(int byteval) throws IOException: ghi một byte xác định bởi tham số *byteval* xuống file, và ném ra IOException khi có lỗi ghi.
- Đóng file: dùng phương thức *close()*

void close( ) throws IOException: sau khi làm việc xong cần đóng file để giải phóng tài nguyên hệ thống đã cấp phát cho file.

Ví dụ: copy nội dung một file text đến một file text khác.

```
/* Copy nội dung của một file text*/
import java.io.*;
class CopyFile
{
    public static void main(String args[])throws IOException
    {
        int i;
        FileInputStream fin;
        FileOutputStream fout;
        try
        {
            // open input file
            try
            {
                fin = new FileInputStream("D:\\source.txt");
            }
            catch(FileNotFoundException exc)
            {
                System.out.println("Input File Not Found");
                return;
            }

            // open output file
            try
            {
                fout = new FileOutputStream("D:\\dest.txt");
            }
            catch(FileNotFoundException exc)
            {
```

```
                System.out.println("Error Opening Output
                File");
                return;
            }
        }
         catch(ArrayIndexOutOfBoundsException exc)
         {
            System.out.println("Usage: CopyFile From To");
            return;
        }

        // Copy File
        try
        {
            do
            {
                i = fin.read();
                if(i != -1) fout.write(i);
            } while(i != -1);
        }
        catch(IOException exc)
        {
            System.out.println("File Error");
        }
        fin.close();
        fout.close();
    }
}
```

Kết quả thực thi chương trình: chương trình sẽ copy nội dung của file D:\source.txt và ghi vào một file mới D:\dest.txt.

5.3.4.Đọc và ghi dữ liệu nhị phân

Phần trên chúng ta đã đọc và ghi các bytes dữ liệu là các ký tự mã ASCII. Để đọc và ghi những giá trị nhị phân của các kiểu dữ liệu trong java, chúng ta sử dụng *DataInputStream* và *DataOutputStream*.

DataOutputStream*:* hiện thực interface *DataOuput*. Interface *DataOutput* có các phương thức cho phép ghi tất cả những kiểu dữ liệu cơ sở của java đến luồng (theo định dạng nhị phân).

Phương thức	Ý nghĩa
void writeBoolean (boolean val)	Ghi xuống luồng một giá trị boolean được xác định bởi val.
void writeByte (int val)	Ghi xuống luồng một byte được xác định bởi val.
void writeChar (int val)	Ghi xuống luồng một Char được xác định bởi val.
void writeDouble (double val)	Ghi xuống luồng một giá trị Double được xác định bởi val.
void writeFloat (float val)	Ghi xuống luồng một giá trị float được xác định bởi val.
void writeInt (int val)	Ghi xuống luồng một giá trị int được xác định bởi val.
void writeLong (long val)	Ghi xuống luồng một giá trị long được xác định bởi val.
void writeShort (int val)	Ghi xuống luồng một giá trị short được xác định bởi val.

Contructor**:** *DataOutputStream(OutputStream outputStream)*

OutputStream: là luồng xuất dữ liệu. Để ghi dữ liệu ra file thì đối tượng *outputStream* có thể là *FileOutputStream*.

DataInputStream: hiện thực interface *DataInput*. Interface *DataInput* có các phương thức cho phép đọc tất cả những kiểu dữ liệu cơ sở của java (theo định dạng nhị phân).

Phương thức	´ nghĩa
boolean readBoolean( )	Đọc một giá trị boolean
Byte readByte( )	Đọc một byte
char readChar( )	Đọc một Char
double readDouble( )	Đọc một giá trị Double
float readFloat( )	Đọc một giá trị float
int readInt( )	Đọc một giá trị int
Long readLong( )	Đọc một giá trị long
short readShort( )	Đọc một giá trị short

Contructor: DataInputStream(InputStream inputStream)

InputStream: là luồng nhập dữ liệu. Để đọ dữ liệu từ file thì đối tượng *InputStream* có thể là *FileInputStream.*

Ví dụ: dùng DataOutputStream và DataInputStream để ghi và đọc những kiểu dữ liệu khác nhau trên file.

```
import java.io.*;
class RWData
{
    public static void main(String args[]) throws IOException
    {
        DataOutputStream dataOut;
        DataInputStream dataIn;
        int i = 10;
        double d = 1023.56;
        boolean b = true;
        try
        {
            dataOut = new DataOutputStream(
                        new FileOutputStream("D:\\testdata"));
        }
        catch(IOException exc)
```

```
{
    System.out.println("Cannot open file.");
    return;
}

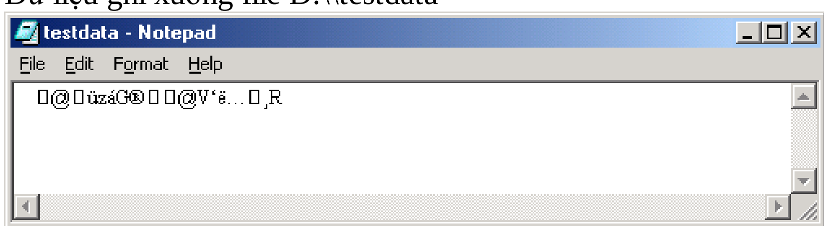
try
{
    System.out.println("Writing " + i);
    dataOut.writeInt(i);
    System.out.println("Writing " + d);
    dataOut.writeDouble(d);
    System.out.println("Writing " + b);
    dataOut.writeBoolean(b);
    System.out.println("Writing " + 12.2 * 7.4);
    dataOut.writeDouble(12.2 * 7.4);
}
catch(IOException exc)
{
    System.out.println("Write error.");
}

dataOut.close();
System.out.println();

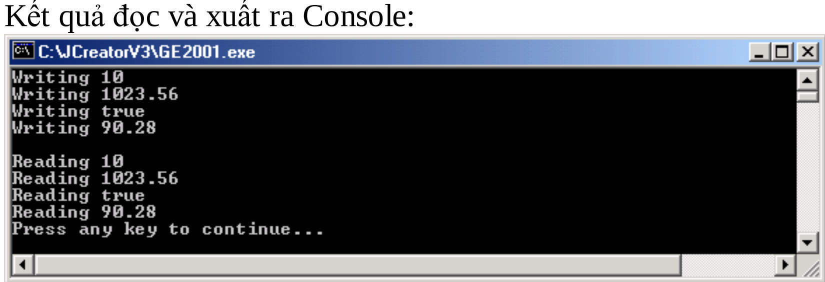
// Now, read them back.
try
{
    dataIn = new DataInputStream(
        new FileInputStream("D:\\testdata"));
}
catch(IOException exc)
{
    System.out.println("Cannot open file.");
    return;
}
```

```
        try
        {
            i = dataIn.readInt();
            System.out.println("Reading " + i);
            d = dataIn.readDouble();
            System.out.println("Reading " + d);
            b = dataIn.readBoolean();
            System.out.println("Reading " + b);
            d = dataIn.readDouble();
            System.out.println("Reading " + d);
        }
        catch(IOException exc)
        {   System.out.println("Read error.");
        }
        dataIn.close();
    }
}
```

Kết quả thực thi chương trình:
Dữ liệu ghi xuống file D:\\testdata



Kết quả đọc và xuất ra Console:



5.4.File truy cập ngẫu nhiên (Random Access Files)

Bên cạnh việc xử lý xuất nhập trên file theo kiểu tuần tự thông qua các luồng, java cũng hỗ trợ truy cập ngẫu nhiên nội dung của một file nào đó dùng *RandomAccessFile*. *RandomAccessFile* không dẫn xuất từ InputStream hay OutputStream mà nó hiện thực các interface *DataInput*, *DataOutput* (có định nghĩa các phương thức I/O cơ bản). *RandomAccessFile* hỗ trợ vấn đề định vị con trỏ file bên trong một file dùng phương thức *seek(long newPos)*.

Ví dụ: minh họa việc truy cập ngẫu nhiên trên file. Chương trình ghi 6 số kiểu double xuống file, rồi đọc lên theo thứ tự ngẫu nhiên.

```
import java.io.*;
class RandomAccessDemo
{
    public static void main(String args[]) throws IOException
    {
        double data[] = {19.4, 10.1, 123.54, 33.0, 87.9, 74.25};
        double d;
        RandomAccessFile raf;

        try
        {
            raf = new RandomAccessFile("D:\\random.dat",
            "rw");
        }
        catch(FileNotFoundException exc)
        {
            System.out.println("Cannot open file.");
            return ;
        }
```

```
// Write values to the file.
for(int i=0; i < data.length; i++)
{
    try
    {
        raf.writeDouble(data[i]);
    }
    catch(IOException exc)
    {
        System.out.println("Error writing to file.");
        return ;
    }
}

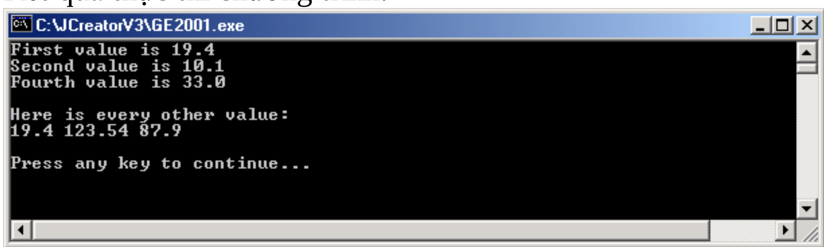
try
{
    // Now, read back specific values
    raf.seek(0); // seek to first double
    d = raf.readDouble();
    System.out.println("First value is " + d);
    raf.seek(8); // seek to second double
    d = raf.readDouble();
    System.out.println("Second value is " + d);
    raf.seek(8 * 3); // seek to fourth double
    d = raf.readDouble();
    System.out.println("Fourth value is " + d);
    System.out.println();

    // Now, read every other value.
    System.out.println("Here is every other value: ");
    for(int i=0; i < data.length; i+=2)
    {   raf.seek(8 * i); // seek to ith double
        d = raf.readDouble();
        System.out.print(d + " ");
    }
```

```
            System.out.println("\n");
        }
        catch(IOException exc)
        {
            System.out.println("Error seeking or reading.");
        }

        raf.close();
    }
}
```

Kết quả thực thi chương trình:



5.5. Sử dụng luồng ký tự

Chúng ta đã tìm hiểu và sử dụng luồng byte để xuất/nhập dữ liệu. Tuy có thể nhưng trong một số trường hợp luồng byte không phải là cách "lý tưởng" để quản lý xuất nhập dữ liệu kiểu character, vì vậy java đã đưa ra kiểu luồng character phục vụ cho việc xuất nhập dữ liệu kiểu character trên luồng.

Mức trên cùng là hai lớp trừu tượng *Reader* và *Writer*. Lớp *Reader* dùng cho việc nhập dữ liệu của luồng, lớp *Writer* dùng cho việc xuất dữ liệu của luồng. Những lớp dẫn xuất từ *Reader* và *Writer* thao tác trên các luồng ký tự Unicode.

Những phương thức định nghĩa trong lớp trừu tượng
Reader và *Writer*

Phương thức	Ý nghĩa

Reader	
abstract void close( )	Đóng luồng
void mark(int numChars)	Đánh dấu vị trí hiện tại trên luồng
boolean markSupported( )	Kiểm tra xem luồng có hỗ trợ thao tác đánh dấu *mark()* không?
int read( )	Đọc một ký tự
int read(char buffer[ ])	Đọc buffer.length ký tự cho vào buffer
abstract int read(char buffer[ ], int offset, int numChars)	Đọc numChars ký tự cho vào vùng đệm buffer tại vị trí buffer[offset]
boolean ready( )	Kiểm tra xem luồng có đọc được không?
void reset( )	Dời con trỏ nhập đến vị trí đánh dấu trước đó
long skip(long numChars)	Bỏ qua numChars của luồng nhập
Writer	
abstract void close( )	Đóng luồng xuất. Có lỗi ném ra IOException
abstract void flush( )	Dọn dẹp luồng (buffer xuất)
void write(int ch)	Ghi một ký tự
void write(byte buffer[ ])	Ghi một mảng các ký tự
abstract void write(char buffer[ ], int offset, int numChars)	Ghi một phần của mảng ký tự
void write(String str)	Ghi một chuỗi
void write(String str, int offset, int numChars)	Ghi một phần của một chuỗi ký tự

5.5.1.Nhập Console dùng luồng ký tự

Thường thì việc nhập dữ liệu từ Console dùng luồng ký tự thì thuận lợi hơn dùng luồng byte. Lớp tốt nhất để đọc dữ liệu nhập từ Console là lớp *BufferedReader*. Tuy nhiên chúng ta không thể xây dựng một lớp *BufferedReader* trực tiếp từ *System.in*. Thay vào đó chúng ta phải chuyển nó thành một luồng ký tự. Để làm điều này chúng ta dùng *InputStreamReader* chuyển bytes thành ký tự.

Để có được một đối tượng *InputStreamReader* gắn với System.in ta dùng constructor của *InputStreamReader*.

```
InputStreamReader(InputStream inputStream)
```

Tiếp theo dùng đối tượng *InputStreamReader* đã tạo ra để tạo ra một *BufferedReader* dùng constructor *BufferedReader*.

```
BufferedReader(Reader inputReader)
```

Ví dụ: Tạo một BufferedReader gắn với Keyboard

```
BufferedReader br = new BufferedReader(new
        InputStreamReader(System.in));
```

Sau khi thực hiện câu lệnh trên, *br* là một luồng ký tự gắn với Console thông qua *System.in*.

Ví dụ: Dùng BufferedReader đọc từng ký tự từ Console. Việc đọc kết thúc khi gặp dấu chấm (dấu chấm để kết thúc chương trình).

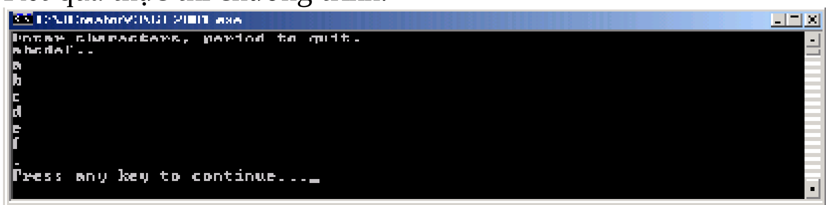
```
import java.io.*;
class ReadChars
{
    public static void main(String args[]) throws IOException
    {
        char c;
        BufferedReader br = newBufferedReader(
```

```
                    new InputStreamReader(System.in));

        System.out.println("Nhap chuoi ky tu,
                                gioi han dau cham.");

        // read characters
        do
        {
            c = (char) br.read();
            System.out.println(c);
        } while(c != '.');
    }
}
```

Kết quả thực thi chương trình:

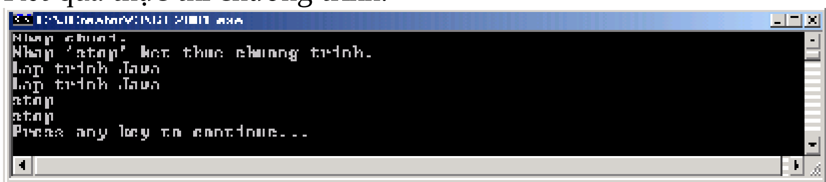


Ví dụ: Dùng BufferedReader đọc chuỗi ký tự từ Console. Chương trình kết thúc khi gặp chuỗi đọc là chuỗi “stop”

```
import java.io.*;
class ReadLines
{
    public static void main(String args[]) throws IOException
    {
        // create a BufferedReader using System.in
        BufferedReader br = new BufferedReader(new
                                InputStreamReader(System.in));
        String str;
        System.out.println("Nhap chuoi.");
        System.out.println("Nhap 'stop' ket thuc chuong trinh.");
```

```
        do
        {
                str = br.readLine();
                System.out.println(str);
        } while(!str.equals("stop"));
    }
}
```

Kết quả thực thi chương trình:



5.5.2.Xuất Console dùng luồng ký tự

Trong ngôn ngữ java, bên cạnh việc dùng *System.out* để xuất dữ liệu ra Console (thường dùng để debug chương trình), chúng ta có thể dùng luồng *PrintWriter* đối với các chương trình "chuyên nghiệp". *PrintWriter* là một trong những lớp luồng ký tự. Việc dùng các lớp luồng ký tự để xuất dữ liệu ra Console thường được "ưa chuộng" hơn.

Để xuất dữ liệu ra Console dùng *PrintWriter* cần thiết phải chỉ định *System.out* cho luồng xuất.

Ví dụ: Tạo đối tượng *PrintWriter* để xuất dữ liệu ra Console

```
        PrintWriter pw = new PrintWriter(System.out, true);
```

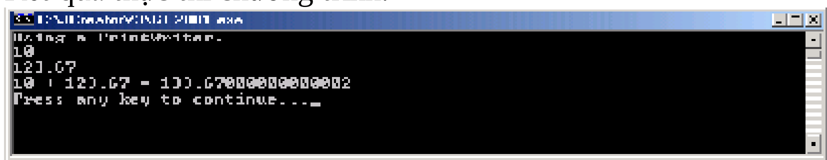
Ví dụ: minh họa dùng *PrintWriter* để xuất dữ liệu ra Console

```
import java.io.*;
public class PrintWriterDemo
{
    public static void main(String args[])
    {
        PrintWriter pw = new PrintWriter(System.out, true);
        int i = 10;
```

```
        double d = 123.67;
        double r = i+d

        pw.println("Using a PrintWriter.");
        pw.println(i);
        pw.println(d);
        pw.println(i + " + " + d + " = " + r);
    }
}
```

Kết quả thực thi chương trình:



5.5.3.Đọc/ghi File dùng luồng ký tự

Thông thường để đọc/ghi file người ta thường dùng luồng byte, nhưng đối với luồng ký tự chúng ta cũng có thể thực hiện được. Ưu điểm của việc dùng luồng ký tự là chúng thao tác trực tiếp trên các ký tự Unicode. Vì vậy luồng ký tự là chọn lựa tốt nhất khi cần lưu những văn bản Unicode.

Hai lớp luồng thường dùng cho việc đọc/ghi dữ liệu ký tự xuống file là *FileReader* và *FileWriter*.

Ví dụ: Đọc những dòng văn bản nhập từ bàn phím và ghi chúng xuống file tên là “test.txt”. Việc đọc và ghi kết thúc khi người dùng nhập vào chuỗi “stop”.

```
import java.io.*;
class KtoD
{
    public static void main(String args[]) throws IOException
    {
        String str;
        FileWriter fw;
```

```
        BufferedReader br = new BufferedReader(
                new InputStreamReader(System.in));

        try
        {
            fw = new FileWriter("D:\\test.txt");
        }
        catch(IOException exc)
        {
            System.out.println("Khong the mo file.");
            return ;
        }

        System.out.println("Nhap ('stop' de ket thuc chuong
        trinh).");

        do
        {
            System.out.print(": ");
            str = br.readLine();
            if(str.compareTo("stop") == 0) break;
            str = str + "\r\n";
            fw.write(str);
        } while(str.compareTo("stop") != 0);

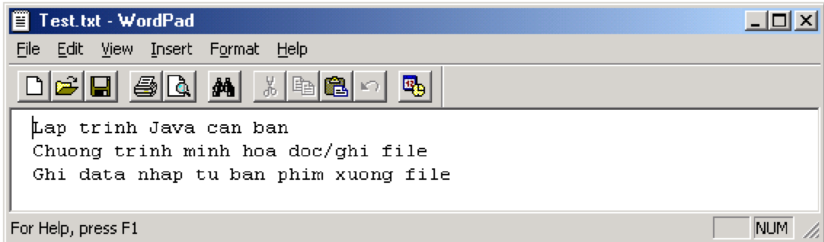
        fw.close();
    }
}
```

Kết quả thực thi chương trình

Dữ liệu nhập từ Console:



Dữ liệu ghi xuống file:



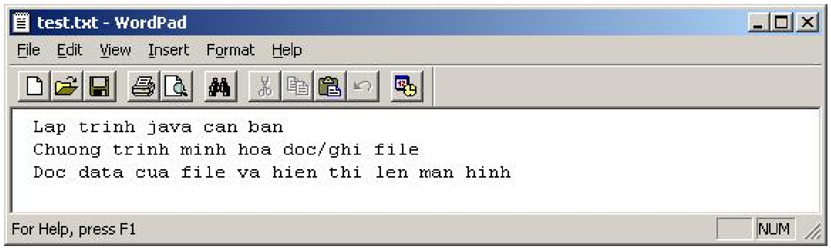
Ví dụ: đọc và hiển thị nội dung của file "test.txt" lên màn hình.

```
import java.io.*;
class DtoS
{
    public static void main(String args[]) throws Exception
    {
        FileReader fr = new FileReader("D:\\test.txt");
        BufferedReader br = new BufferedReader(fr);
        String s;
        while((s = br.readLine()) != null)
        {
            System.out.println(s);
        }

        fr.close();
    }
}
```

Kết quả thực thi chương trình

Nội dung của file test.txt:



Kết quả đọc file và hiển thị ra Console:



5.6.Lớp File

Lớp File không phục vụ cho việc nhập/xuất dữ liệu trên luồng. Lớp File thường được dùng để biết được các thông tin chi tiết về tập tin cũng như thư mục (tên, ngày giờ tạo, kích thước, …)

java.lang.Object
  +--**java.io.File**

Các Constructor:

Tạo đối tượng File từ đường dẫn tuyệt đối
*public **File**(String pathname)*
ví dụ: *File f = new File("C:\\Java\\vd1.java");*

Tạo đối tượng File từ tên đường dẫn và tên tập tin tách biệt
*public **File**(String parent, String child)*
ví dụ: *File f = new File("C:\\Java", "vd1.java");*

Tạo đối tượng File từ một đối tượng File khác
*public **File**(File parent, String child)*
ví dụ: *File dir = new File ("C:\\Java");*
File f = new File(dir, "vd1.java");

Một số phương thức thường gặp của lớp File (chi tiết về các phương thức đọc thêm trong tài liệu J2SE API Specification)

*public String **getName()***	Lấy tên của đối tượng File
*public String **getPath()***	Lấy đường dẫn của tập tin
*public boolean **isDirectory()***	Kiểm tra xem tập tin có phải là thư mục không?
*public boolean **isFile()***	Kiểm tra xem tập tn có phải là một file không?
…	
*public String[] **list()***	Lấy danh sách tên các tập tin và thư mục con của đối tượng File đang xét và trả về trong một mảng.

Ví dụ:

```
import java.awt.*;
import java.io.*;
public class FileDemo
{
    public static void main(String args[])
    {
        Frame fr = new Frame ("File Demo");
        fr.setBounds(10, 10, 300, 200);
        fr.setLayout(new BorderLayout());

        Panel p = new Panel(new GridLayout(1,2));

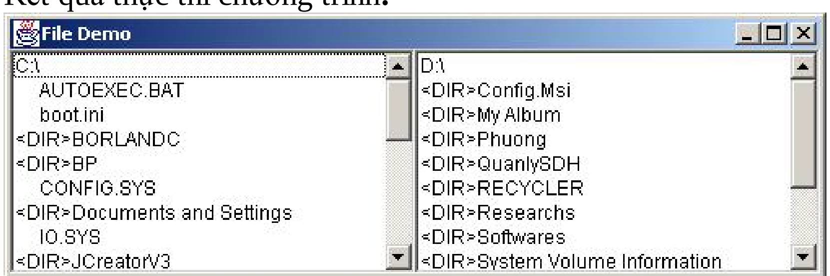
        List list_C = new List();
        list_C.add("C:\\");
        File driver_C = new File ("C:\\");
        String[] dirs_C = driver_C.list();
```

```
        for (int i=0;i<dirs_C.length;i++)
        {   File f = new File ("C:\\" + dirs_C[i]);
            if (f.isDirectory())
                list_C.add("<DIR>" + dirs_C[i]);
            else
                list_C.add("     " + dirs_C[i]);
        }

        List list_D = new List();
        list_D.add("D:\\");
        File driver_D = new File ("D:\\");
        String[] dirs_D = driver_D.list();
        for (int i=0;i<dirs_D.length;i++)
        {   File f = new File ("D:\\" + dirs_D[i]);
            if (f.isDirectory())
                list_D.add("<DIR>" + dirs_D[i]);
            else
                list_D.add("     " + dirs_D[i]);
        }

        p.add(list_C);
        p.add(list_D);
        fr.add(p, BorderLayout.CENTER);
        fr.setVisible(true);
    }
}
```

Kết quả thực thi chương trình**:**



Chương 6: LẬP TRÌNH CƠ SỞ DỮ LIỆU

6.1.GIỚI THIỆU

Hầu hết các chương trình máy tính hiện này đếu ít nhiều liên quan đến việc truy xuất thông tin trong các cơ sở dữ liệu. Chính vì thế nên các thao tác hỗ trợ lập trình cơ sở dữ liệu là chức năng không thể thiếu của các ngôn ngữ lập trình hiện đại, trong đó có Java. JDBC API là thư viện chứa các lớp và giao diện hỗ trợ lập trình viên Java kết nối và truy cập đến các hệ cơ sở dữ liệu.

Phiên bản JDBC API mới nhất hiện nay là 3.0, là một thành phần trong J2SE, nằm trong 2 gói thư viện:

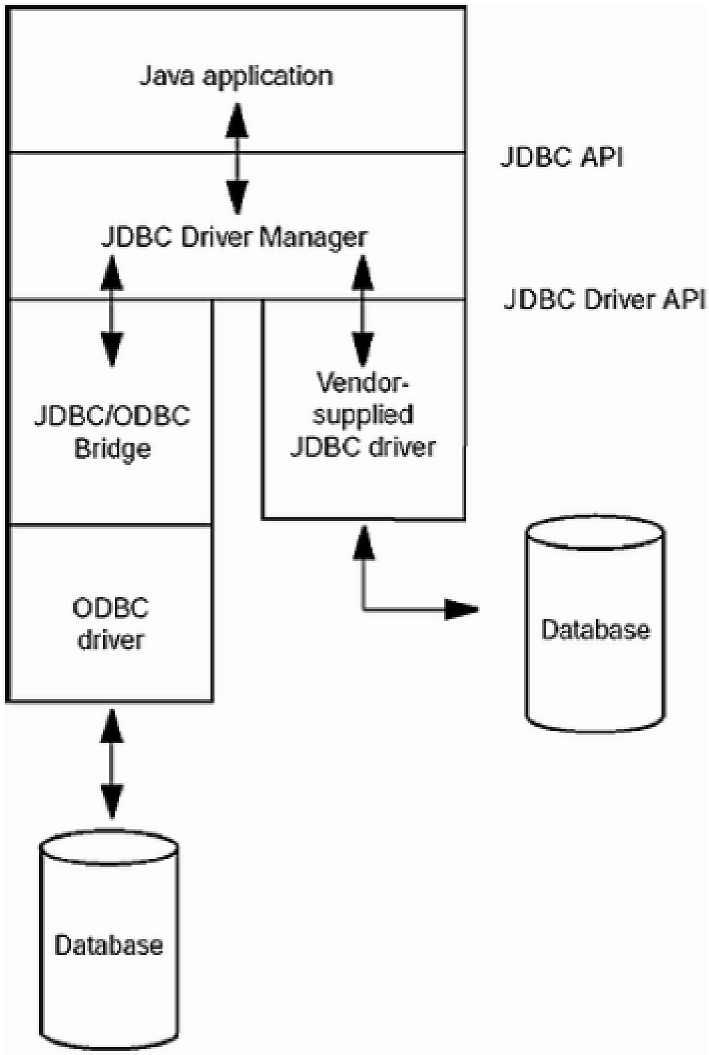
- java.sql: chứa các lớp và giao diên cơ sở của JDBC API.
- javax.sql: chứa các lớp và giao diện mở rộng.

JDBC API cung cấp cơ chế cho phép một chương trình viết bằng Java có khả năng độc lập với các hệ cơ sở dữ liệu, có khả năng truy cập đến các hệ cơ sở dữ liệu khác nhau mà không cần viết lại chương trình. JDBC đơn giản hóa việc tạo và thi hành các câu truy vấn SQL trong chương trình.

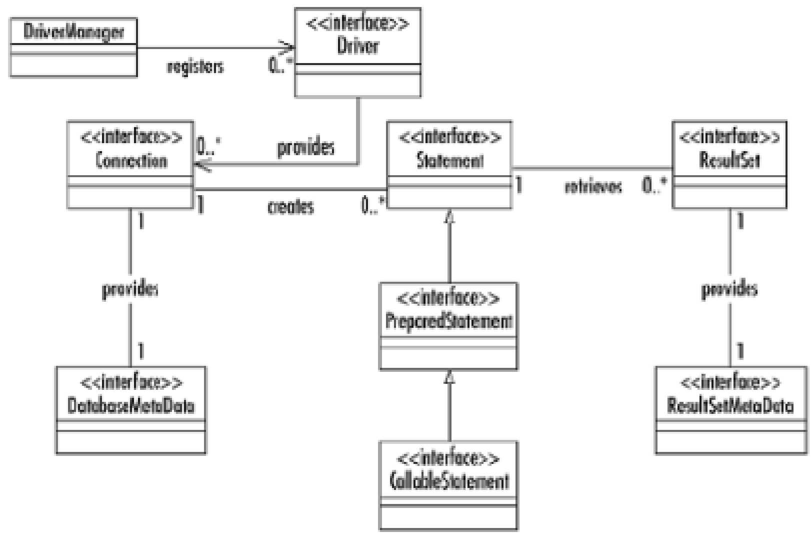
6.2.KIẾN TRÚC JDBC

Kiến trúc của của JDBC tương tự như kiến trúc ODBC do Microsoft xây dựng. Theo kiến trúc này các thao tác liên quan đến cơ sở dữ liệu trong chương trình được thực hiện thông qua các JDBC API. Sau đó các JDBC API sẽ truyền các yêu cầu của chương trình đến bộ quản lý trình điều khiển JDBC, là bộ phận có nhiệm vụ lựa chọn trình điều khiển thích hợp để có thể làm việc với cơ sở dữ liệu cụ thể mà chương trình muốn kết nối.

Như vậy kiến trúc của JDBC gồm 2 tầng: tầng đầu tiên là các JDBC API, có nhiệm vụ chuyển các câu lệnh SQL cho bộ quản lý trình điều khiển JDBC; tầng thứ 2 là các JDBC Driver API, thực hiện nhiệm vụ liện hệ vớ trình điều khiển của hệ quản trỉ cơ sở dữ liệu cụ thể.



Hình bên dưới minh họa các lớp và giao diện cơ bản trong JDBC API.



6.3.Các khái niệm cơ bản

6.3.1.JDBC Driver

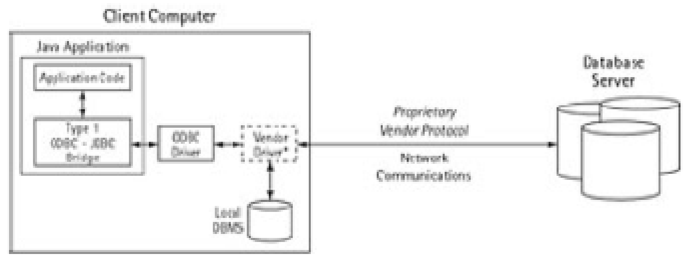
Để có thể tiến hành truy cập đến các hệ quản trị cơ sở dữ liệu sử dụng kỹ thuật JDBC, chúng ta cần phải cò trình điều khiển JDBC của hệ quản trị CSDL mà chúng ta đang sử dụng. Trình điều khiển JDBC là đoạn chương trình, do chính nhà xây dựng hệ quản trị CSDL hoặc do nhà cung ứng thứ ba cung cấp, có khả năng yêu cầu hệ quản trị CSDL cụ thể thực hiện các câu lệnh SQL.

Danh sách các trình điều khiển JDBC cho các hệ quản trị CSDL khác nhau được Sun cung cấp và cập nhật liên tục tại địa chỉ: *http://industry.java.sun.com/products/jdbc/drivers*.

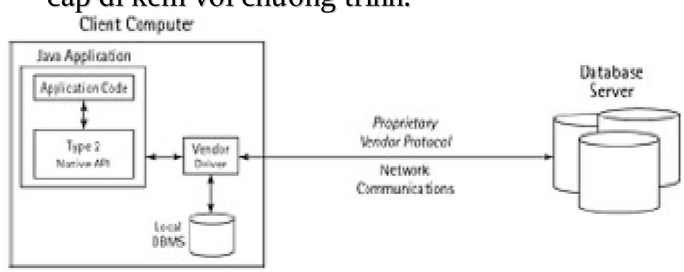
Các trình điều khiển JDBC được phân làm 04 loại khác nhau.

- Loại 1: có tên gọi là ***Bridge Driver***. Trình điều khiển loại này kết nối với các hệ CSDL thông qua cầu nối ODBC. Đây chính là chình điều khiển

được sử dụng phổ biến nhất trong những ngày đầu Java xuất hiện. Tuy nhiên, ngày nay trình điều khiển loại này không còn phổ biến do có nhiều hạn chế. Trình điều khiển loại này luôn được cung cấp kèm trong bộ J2SE với tên: sun.jdbc.odbc.JdbcOdbcDriver.

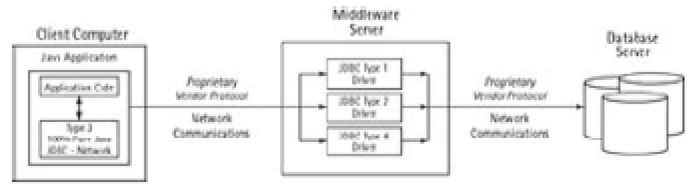


- Loại 2: có tên gọi là ***Native API Driver***. Trình điều khiển loại này sẽ chuyển các lời gọi của JDBC API sang thư viện hàm (API) tương ứng với từng hệ CSDL cụ thể. Trình điều khiện loại này thường chỉ do nhà xây dựng hệ CSDL cung cấp. Để có thể thi hành chương trình mã lệnh để làm việc với hệ CSDL cụ thể cần phải được cung cấp đi kèm với chương trình.

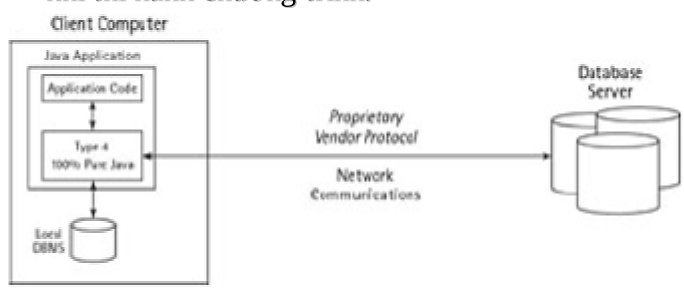


- Loại 3: có tên gọi là ***JDBC-Net Driver***. Trình điều khiển loại này sẽ chuyển các lời gọi JDBC API sang một dạng chuẩn độc lập với các hệ CSDL, và sau được chuyển sang lời gọi của hệ CSDL cụ thể bỡi 1 chương trình trung gian. Trình điều khiển của các nhà cung ứng thứ 3 thường thuộc loại này. Lợi thế của trình điều khiển loại

này là không cần cung cấp mã lệnh kèm theo và có thể sử dụng cùng một trình điều khiển để truy cập đến nhiều hệ CSDL khác nhau.



- Loại 4: có tên gọi là ***Native Protocol Driver***. Trình điều khiển loại này chuyển các lời gọi JDBC API sang mã lệnh của hệ CSDL cụ thể. Đây là các trình điều khiển thần Java, có nghĩa là không cần phải có mã lệnh của hệ CSDL cụ thể khi thi hành chương trình.



6.3.2.JDBC URL

Để có thể kết nối với CSDL, chúng ta cần xác định nguồn dữ liệu cùng với các thông số liên quan dưới dạng 1 URL như sau:
jdbc:<subprotocol>:<dsn>:<others>
Trong đó:

- ***<subprotocol>***: được dùng để xác định trình điều khiển để kết nối với CSDL.
- ***<dsn>***: địa chỉ CSDL. Cú pháp của ***<dsn>*** phụ thuộc vào từng trình điều khiển cụ thể.
- ***<other>***: các tham số khác

Ví dụ:

- jdbc:odbc:dbname là URL để kết nối với CSDL tên dbname sử dụng cầu nối ODBC.
- jdbc:microsoft:sqlserver://hostname:1433 là URL để kết nối với CSDL Microsoft SQL Server. Trong đó hostname là tên máy cài SQL Server.

6.4.KẾT NỐI CSDL VỚI JDBC

Việc kết nối với CSDL bằng JDBC được thực hiện qua hai bước: đăng ký trình điều khiển JDBC; tiếp theo thực thi phương thức getConnection() của lớp DriverManager.

6.4.1.Đăng ký trình điều khiển

Trình điều khiển JDBC được nạp khi mã bytecode của nó được nạp vào JVM. Một cách đơn giản để thực hiện công việc này là thực thi phương thức Class.forName("*<JDBC Driver>*").
Ví dụ: để nạp trình điều khiển sử dụng cầu nối ODBC do Sun cung cấp, chúng ta sử dụng câu lệnh sau Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").

6.4.2.Thực hiện kết nối

Sau khi đã nạp trình điều khiển JDBC, việc kết nối với CSDL được thực hiện với một trong các phương thức sau trong lớp DriverManager:

- public static Connection getConnection(String url) throws SQLException: thực hiện kết nối với CSDL được yêu cầu. Bộ quản lý trình điều khiển sẽ tự động lựa chọn trình điều khiển phù hợp trong số các trình điều khiển đã được nạp.
- public static Connection getConnection(String url, String user, String pass) throws SQLException: tiến hành kết nối tới CSDL với tài khoản user và mật mã pass.
- public static Connection getConnection(String url, Properties info) throws SQLException: tương tự hai phương thức trên ngoài ra cung cấp thêm các

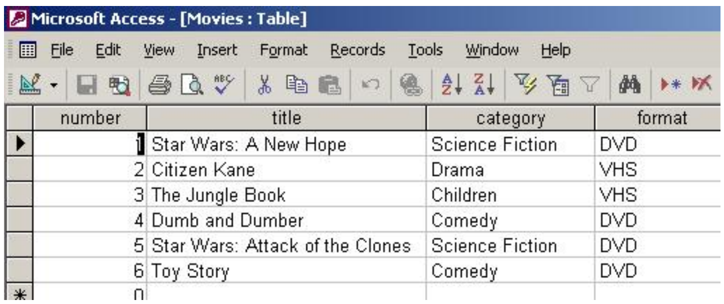
thông tin qui định thuộc tính kết nối thông qua đối tượng của lớp Properties.

Kết quả trả về của các phương thức trên là một đối tượng của lớp java.sql.Connection được dùng để đại diện cho kết nối đến CSDL.

6.4.3.Ví dụ

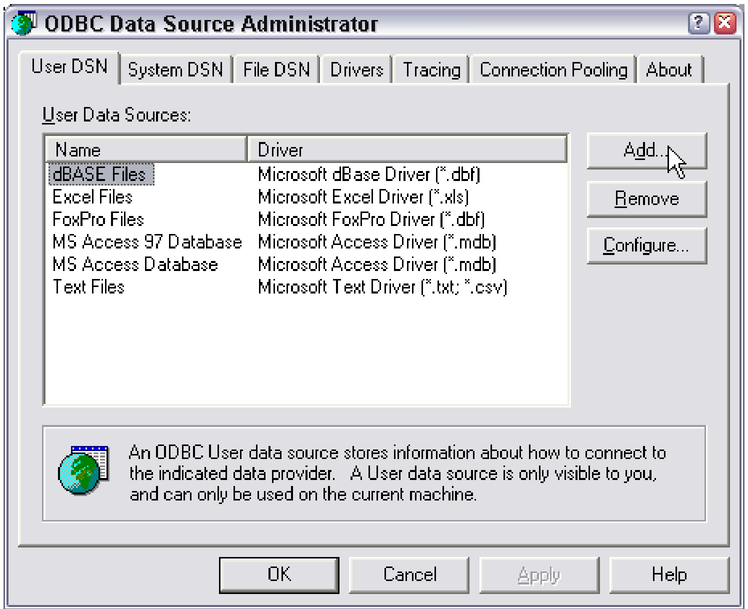
Trong phần ví dụ này chúng ta sẽ tìm hiểu các cách khác nhau để kết nối với tập tin CSDl Access movies.mdb có một bảng tên Movies. Bảng này gồm các cột number, title, category và fomat.

Để có thể tiến hành kết nối với Microsoft Access thông qua cầu

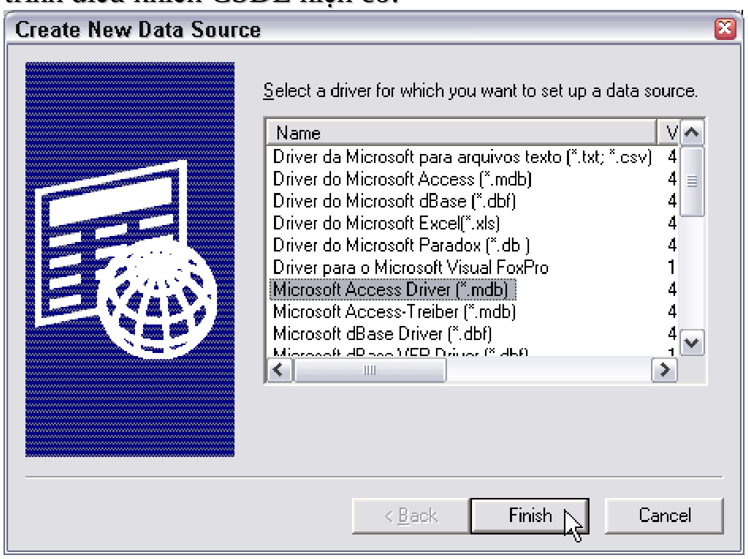


number	title	category	format
1	Star Wars: A New Hope	Science Fiction	DVD
2	Citizen Kane	Drama	VHS
3	The Jungle Book	Children	VHS
4	Dumb and Dumber	Comedy	DVD
5	Star Wars: Attack of the Clones	Science Fiction	DVD
6	Toy Story	Comedy	DVD
0			

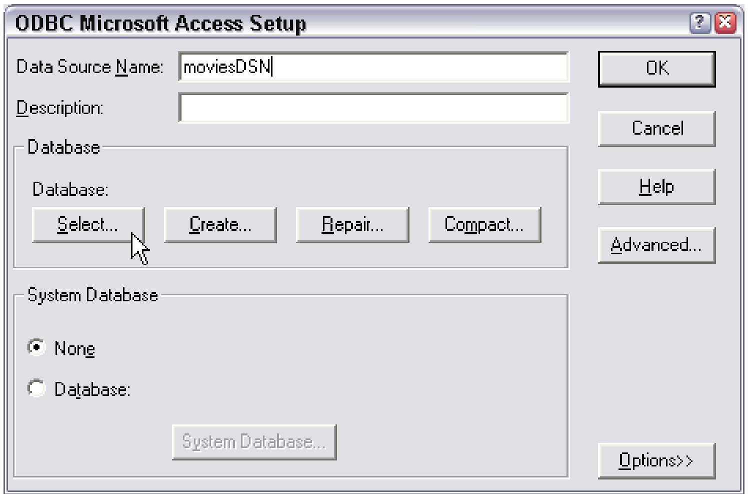
nối ODBC sau khi đã tạo tập tin CSDL movies.mdb, chúng ta cần phải tạo Data Source Name cho CSDL bằng cách vào Control Panel và chọn ODBC Data Source.



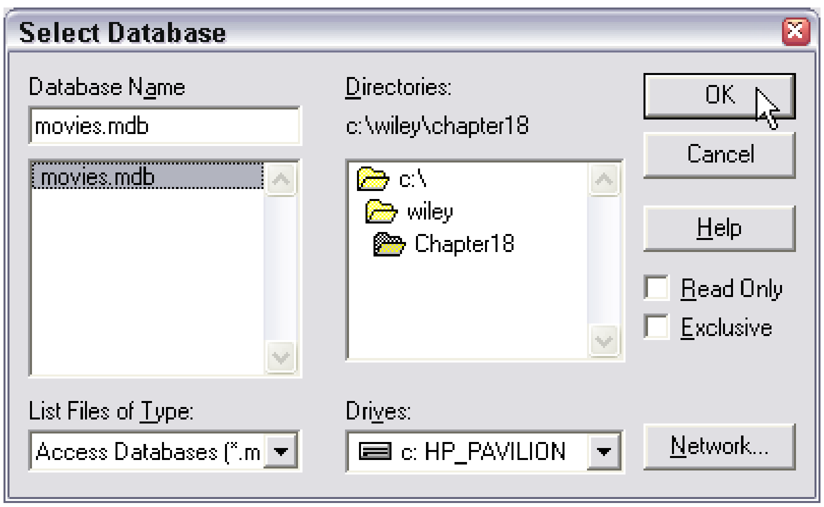
Tiếp theo nhấn vào nút Add, bạn sẽ thấy hiển thị danh sách các trình điều khiển CSDL hiện có.



Bạn chọn Microsoft Access Driver(*.mdb) và nhấn Finish. Cửa sổ cấu hình cho tập tin Access sẽ xuất hiện và nhập moviesDSN vào ô Data Source Name



Bạn nhấn nút Select và chọn tập tin CSDL cần tạo data source name. Sau đó nhấn OK để kết thúc.



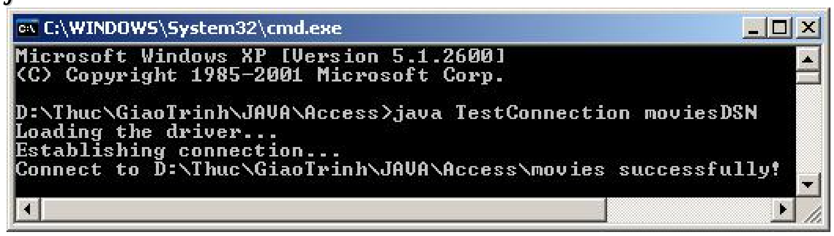
Sau khi đã hoàn tất công việc tạo DSN cho tập tin movies.mdb, chúng ta có thể sử dụng đoạn mã sau để tiến hành kết nối với tập tin movies.mdb.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class TestConnection{
    public static void main(String args[]) {
        Connection connection = null;
        if( args.length != 1) {
            System.out.println("Syntax: java TestConnection " +
                                            "DSN");
            return;
        }
        try { // load driver
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println("Loading the driver...");
        }
        catch( Exception e ) { //problem load driver,class not
exist
            e.printStackTrace( );
            return;
        }
        try {
            String dbURL = "jdbc:odbc:" + args[0];
            System.out.println("Establishing connection...");
            connection =
                    DriverManager.getConnection(dbURL,"","");
            System.out.println("Connect to ” +
                    connection.getCatalog() + “ successfully!");
            // Do whatever queries or updates you want here!!!
        }
        catch( SQLException e ) {
            e.printStackTrace( );
        }
```

```
    finally {
        if( connection != null ) {
            try { connection.close( ); }
            catch( SQLException e ) {
                e.printStackTrace( );
            }
        }
    }
  }
}
```

Sau khi biên dịch đoạn chương trình trên, chúng ta có thể thực hiện kết nối với CSDL bằng cách thực thi câu lệnh:

java TestConnection moviesDSN



6.5.KIỂU DỮ LIỆU SQL VÀ KIỂU DỮ LIỆU JAVA

Trong quá trình thao tác với CSDL, chúng ta sẽ gặp phải vấn đề chuyển đổi giữa kiểu dữ liệu trong CSDL sang kiểu dữ liệu Java hỗ trợ và ngược lại. Việc chuyển đổi này được thực hiện như trong 2 bảng sau.

SQL Type	Java Type
BIT	Boolean
TINYINT	Byte
SMALLINT	Short
INTEGER	Int
BIGINT	Long
REAL	Float
FLOAT	Double
DOUBLE	Double

DECIMAL	java.math.BigDecimal
NUMERIC	java.math.BigDecimal
CHAR	java.lang.String
VARCHAR	java.lang.String
LONGVARCHAR	java.lang.String
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
BINARY	byte[ ]
VARBINARY	byte[ ]
LONGVARBINARY	byte[ ]
BLOB	java.sql.Blob
CLOB	Java.sql.Clob
ARRAY	Java.sql.Array
REF	Java.sql.Ref
STRUCT	Java.sql.Struct

Bảng chuyển đổi từ kiểu dữ liệu SQL sang Java

Java Type	SQL Type
boolean	BIT
byte	TINYINT
short	SMALLINT
int	INTEGER
long	BIGINT
float	REAL
double	DOUBLE
java.math.BigDecimal	NUMERIC
java.lang.String	VARCHAR or LONGVARCHAR
byte[ ]	VARBINARY or LONGVARBINARY
java.sql.Date	DATE
java.sql.Time	TIME
java.sql.Timestamp	TIMESTAMP
java.sql.Blob	BLOB
java.sql.Clob	CLOB
java.sql.Array	ARRAY
java.sql.Ref	REF

java.sql.Struct	STRUCT

Bảng chuyển đổi từ kiểu dữ liệu Java sang SQL

6.6.CÁC THAO TÁC CƠ BẢN TRÊN CSDL

Các thao tác truy vấn CSDL chỉ có thể được thực hiện sau khi đã có đối tượng Connection, được tạo ra từ quá trình kết nối vào CSDL. Chúng ta sử dụng đối tượng của lớp Connection để tạo ra các thể hiện của lớp java.sql.Statement. Sau khi tạo ra các đối tượng của lớp Statement chúng ta có thể thực hiện các thao tác trong các đối tượng statement trên connection tương ứng.
Nội dung trong một statement chính là các câu SQL. Câu lệnh SQL trong các statement chỉ được thực hiện khi chúng ta gửi chúng đến CSDL. Nếu câu lện SQL là một câu truy vấn nội dung thì kết quả trả về sẽ là một thể hiện của lớp java.sql.ResultSet, ngược lại (các câu lệnh thay đổi nội dung CSDL) sẽ trả về kết quả là mộ số nguyên. Các đối tượng của lớp ResultSet cho phép chúng ta truy cập đến kết quả trả về của các câu truy vấn.

6.6.1.Các lớp cơ bản

- § java.sql.Statement
  Statement là một trong 3 lớp JDBC cơ bản dùng để thể hiện một câu lệnh SQL. Mọi thao tác trên CSDL được thực hiện thông qua 3 phương thức của lớp Statement.
  Phương thức executeQuery() nhận vào 1 tham số là chuỗi nội dung câu lện SQL và trả về 1 đối tượng kiểu ResultSet. Phương thức này được sử dụng trong các trường hợp câu lệnh SQL có trả về các kết quả trong CSDL.
  Phương thức executeUpdate() cũng nhận vào 1 tham số là chuỗi nội dung câu lệnh SQL. Tuy nhiện phương thức này chỉ sử dụng được đối với các cây lệnh cập nhật nội dung CSDL. Kết quả trả về là số dòng bị tác động bỡi câu lệnh SQL.

Phương thức execute() là trường hợp tổng quát của 2 phương thức trên. Phương thức nhận vào chuỗi nội dung câu lệnh SQL. Câu lệnh SQL có thể là câu lệnh truy vấn hoặc cập nhật. Nếu kết quả của câu lệnh là các dòng trong CSDL thì phương thức trả về giá trị true, ngược lại trả về giá trị false. Trong trường hợp giá trị true, sau đó chúng ta có thể dùng phương thức getResultSet() để lấy các dòng kết quả trả về.

§ java.sql.ResultSet

Đối tượng resultset là các dòng dữ liệu trả về của câu lệnh truy vấn CSDL. Lớp này cung cấp các phương thức để rút trích các cột trong từng dòng kết quả trả về. Tất cả các phương thức này đều có dạng:

type get**Type**(int | String)

Trong đó tham số có thể là số thứ tự của cột hoặc tên cột cần lấy nội dung.

Tại 1 thời điểm chúng ta chỉ có thể thao tác trên 1 dòng của resultset. Để thao tác trên dòng tiếp theo chúng ta sử dụng phương thức next(). Phương thức trả về giá trị true trong trường hợp có dòng tiếp theo, ngược lại trả về giá trị false.

6.6.2.Ví dụ truy vấn CSDL

```
public class Movie{
   private String movieTitle, category, mediaFormat;
   private int number;

   public Movie(int n, String title, String cat, String format){
      number = n;
      movieTitle = title;
      category = cat;
      mediaFormat = format;
   }
```

```
    public int getNumber(){return number;}

    public String getMovieTitle(){return movieTitle;}

    public String getCategory(){return category;}

    public void setCategory(String c){category = c;}

    public String getFormat(){return mediaFormat;}

    public void setFormat(String f){mediaFormat = f;}

    public String toString(){
        return number + ": " + movieTitle + " - " + category + "
"                                                       + mediaFormat;
    }
}


import java.sql.*;

public class MovieDatabase{
    private Connection connection;
    private PreparedStatement findByNumber, updateCategory;
    private CallableStatement findByCategory;

    public MovieDatabase(Connection connection) throws
                                                        SQLException{
        this.connection = connection;
    }

    public void showAllMovies(){
        try{
            Statement selectAll = connection.createStatement();
            String sql = "SELECT * FROM Movies";
```

```
            ResultSet results =   selectAll.executeQuery(sql);
            while(results.next()){
                int number = results.getInt(1);
                String title = results.getString("title");
                String category = results.getString(3);
                String format = results.getString(4);

                Movie movie = new Movie(number, title, category,
                                                        format);
                System.out.println(movie.toString());
            }
            results.close();
            selectAll.close();
        }
        catch(SQLException e){
            e.printStackTrace();
        }
    }
}

import java.sql.*;

public class ShowMovies{
    public static void main(String [] args){
        String url = "jdbc:odbc:" + args[0];
        try{
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection connection =
                        DriverManager.getConnection(url);
            MovieDatabase          db          =          new
MovieDatabase(connection);
            db.showAllMovies();
            connection.close();
        }
        catch(Exception e){
```

```
            e.printStackTrace();
        }
    }
}
```

6.6.3.Ví dụ cập nhật CSDL

Phương thức addMovie() bên dưới được thêm vào lớp MovieDatabase đã định nghĩa ở ví dụ trên.

```
public class MovieDatabase{
          …
    public void addMovie(Movie movie){
        System.out.println("Adding movie: " + movie.toString());
        try{
            Statement addMovie = connection.createStatement();
            String sql = "INSERT INTO Movies VALUES("
                        + movie.getNumber() + ", "
                        + "'" + movie.getMovieTitle() + "', "
                        + "'" + movie.getCategory() + "', "
                        + "'" + movie.getFormat() + "')";
            System.out.println("Executing statement: " + sql);
            addMovie.executeUpdate(sql);
            addMovie.close();
            System.out.println("Movie added successfully!");
        }
        catch(SQLException e){
            e.printStackTrace();
        }
    }
}

import java.sql.*;

public class AddMovies{
    public static void main(String [] args){
        String url = "jdbc:odbc:" + args[0];
```

```
        System.out.println("Attempting to connect to " + url);
        try{
            System.out.println("Loading the driver...");
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            System.out.println("Establishing a connection...");
            Connection connection =
                        DriverManager.getConnection(url);
            System.out.println("Connect to "
                    + connection.getCatalog() + " a success!");
            MovieDatabase db = new
MovieDatabase(connection);
            Movie [] movies = new Movie[6];
            movies[0] = new Movie(1, "Star Wars: A New Hope",
                                "Science Fiction", "DVD");
            movies[1] = new Movie(2, "Citizen Kane", "Drama",
                                "VHS");
            movies[2] = new Movie(3, "The Jungle Book",
                                "Children", "VHS");
            movies[3] = new Movie(4, "Dumb and Dumber",
                        "Comedy", "DVD");
            movies[4] = new Movie(5, "Star Wars: Attack of the
                        Clones", "Science Fiction", "DVD");
            movies[5] = new Movie(6, "Toy Story", "Children",
                        "DVD");
            for(int i = 0; i < movies.length; i++){
                db.addMovie(movies[i]);
            }
            System.out.println("Closing the connection...");
            connection.close();
        }
        catch(Exception e){
            e.printStackTrace();
        }
    }
}
```

Tài liệu tham khảo:

[1] java.sun.com
[2] Herbert Schildt. **Java 2. A Beginner's Guide**. Second Edition. McGraw-Hill - 2003.
[3] Dr. Harvey M. Deitel - Paul J. Deitel. **Java How to Program**, 4th Ed (Deitel). Prentice Hall - 2002
[4] Simon Roberts – Philip Heller – Michael Ernest. **Complete Java 2 Certification – study guide**. BPB Publications – 2000.
[5] Cay S. Horstmann – Gary Cornell. **Core Java Volum 1 - Fundamentals.** The Sun Microsystems press. 1997
[6] Cay S. Horstmann – Gary Cornell. **Core Java Volum 2 – Advanced Features.** The Sun Microsystems press. 1997

Phụ lục A: Trắc nghiệm kiến thức

1. Chọn phát biểu đúng
a. InputStream và OuputStream là 2 luồng dữ liệu kiểu byte
b. Reader và Writer là 2 luồng dữ liệu kiểu character.
c. Câu a) và b) đúng
d. Tất cả các câu trên đều sai

2. Cho biết số byte mà đoạn chương trình sau ghi ra tập tin temp.txt

```
import java.io.*;

public class TestIOApp {
 public static void main(String args[]) throws IOException {
  FileOutputStream outStream = new FileOutputStream("test.txt");
  String s = "test";
  for(int i=0;i<s.length();++i)
   outStream.write(s.charAt(i));
  outStream.close();
 }
```

a. 2 bytes
b. 4 bytes
c. 8 bytes
d. 16 bytes

3. Chọn phát biểu đúng
a. Một thể hiện của lớp File có thể được dùng để truy cập các tập tin trong thư mục hiện hành
b. Khi một thể hiện của lớp File được tạo ra thì một tập tin tương ứng cũng được tạo ra trên đĩa.
c. Các thể hiện của lớp File được dùng để truy cập đến các tập tin và thư mục trên đĩa
d. Câu a) và c) đúng

4. Cho biết cách tạo một thể hiện của InputStreamReader từ một thể hiện của InputStream.
a. Sử dụng phương thức createReader() của lớp InputStream
b. Sử dụng phương thức createReader() của lớp InputStreamReader
c. Tạo một thể hiện của InputStream rồi truyền vào cho hàm khởi tạo của InputStreamReader
d. Tất cả các câu trên đều sai

5. Chọn phát biểu đúng
a. Lớp Writer có thể được dùng để ghi các ký tự có cách mã hóa khác nhau ra luồng xuất
b. Lớp Writer có thể được dùng để ghi các ký tự Unicode ra luồng xuất
c. Lớp Writer có thể được dùng để ghi giá trị các kiểu dữ liệu cơ sở ra luồng xuất
d. Câu a) và b) đúng

6. Chọn phát biểu đúng:
a. Các event listeners là các interface qui định các phương thức cần phải cài đặt để xử lý các sự kiên liên quan khi sự kiện đó xảy ra.
b. Một event adapter là một cung cấp các cài đặt mặc định cho các event listener tương ứng
c. Lớp WindowAdapter được dùng để xử lý các sự kiện liên quan đến cửa sổ màn hình.
d. Tất cả các câu trên đều đúng

7. Khi có nhiều component được gắn các bộ lắng nghe của cùng một loại sự kiện thì component nào sẽ nhận được sự kiện đầu tiên?
a. Component đầu tiên được gắn bộ lắng nghe sự kiện
b. Component cuối cùng được gắn bộ lắng nghe sự kiện
c. Không thể xác định component nào sẽ nhận trước

d. Không thể có nhiều hơn một bộ lắng nghe cho cùng một loại sự kiện

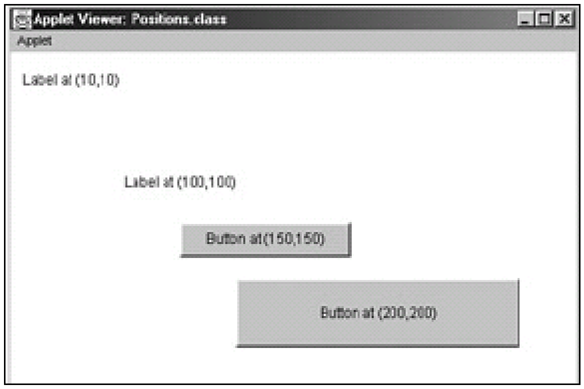
8. Chọn các component có phát sinh action event
a. Button
b. Label
c. Checkbox
d. Windows

9. Chọn phát biểu đúng
a. Thể hiện của TextField có thể phát sinh ActionEvent
b. Thể hiện của TextArea có thể phát sinh ActionEvent
c. Thể hiện của button có thể phát sinh ActionEvent
d. Câu a) và c) đúng

10. Chọn phát biểu đúng
a. MouseListener interface định nghĩa các phương thức để xử lý sự kiện nhấn chuột.
b. MouseMotionListener interface định nghĩa các phương thức để xử lý sự kiện nhấn chuột.
c. MouseClickListener interface định nghĩa các phương thức để xử lý sự kiện nhấn chuột.
d. Tất cả các câu trên đều đúng

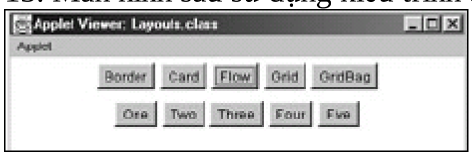
11. Giả sữ chúng ta có thể hiện e của bộ lắng nghe sự kiện TextEvent và thể hiện t của lớp TextArea. Cho biết cách để gắn bộ lắng nghe e vào t?
a. t.addTextListener(e)
b. e.addTextListener(t)
c. addTextListener(e,t)
d. addTextListener(t,e)

12. Màn hình sau sử dụng kiểu trình bày nào?



a. CardLayout
b. nullLayout
c. BorderLayout
d. SetLayout

13. Màn hình sau sử dụng kiểu trình bày nào?



a. GridLayout
b. FlowLayout
c. BorderLayout
d. GridBagLayout

14. Cho một component comp và một container cont có kiểu trình bày là BorderLayout. Cho biết cách để gắn comp vào vị trí đầu của cont.
a. addTop(cont,comp)
b. comp.add(“North”, cont)
c. cont.addTop(comp)
d. cont.add(comp,BorderLayout.NORTH)

15. Cho một component comp và một container cont có kiểu trình bày là FlowLayout. Cho biết cách để gắn comp vào cont.

a. cont.add(comp)
b. comp.add(cont)
c. cont.addComponent(comp)
d. cont.addAllComponents()

16. Chọn phương thức dùng để xác định cách trình bày của một khung chứa
a. startLayout()
b. initLayout()
c. layoutContainer()
d. setLayout()

17. Chọn phương thức dùng để xác định vị trí và kích thước của các component
a. setBounds()
b. setSizeAndPosition()
c. setComponentSize()
d. setComponent()

18. Chọn kiểu trình bày để đặt các component trên khung chứa dưới dạng bảng.
a. CardLayout
b. BorderLayout
c. GridLayout
d. FlowLayout

19. Chọn phương thức dùng để gán nội dung cho Label
a. setText()
b. setLabel()
c. setTextLabel()
d. setLabelText()

20. Chọn phát biểu đúng
a. TextComponent extends TextArea
b. TextArea extends TextField

c. TextField extends TextComponent
d. TextComponent extends TextField

21. Chọn phát biểu đúng
a. Lớp CheckboxGroup dùng để định nghĩa cá RadioButtons
b. Lớp RadioGroup dùng để định nghĩa cá RadioButtons
c. Tất cả các câu trên đều đúng
d. Tất cả các câu trên đều sai

22. Chọn câu lệnh để tạo ra TextArea có 10 dòng và 20 cột
a. new TexArea(10,20)
b. new TexArea(20,10)
c. new TexArea(200)
d. Tất cả các câu trên đều sai

23. Chọn câu lệnh để tạo ra một danh sách gồm 5 mục chọn và cho phép thực hiện chọn nhiều mục cùng lúc
a. new List(5, true)
b. new List(true, 5)
c. new List(5, false)
d. new List(false, 5)

24. Chọn phương thức để hiện thị Frame lên màn hình
a. setVisible()
b. display()
c. displayFrame()
d. Tất cả các câu trên đều sai

25. Chọn phát biểu đúng
a. Lớp Class là lớp cha của lớp Object
b. Lớp Object là một lớp final
c. Mọi lớp đề kế thừa trực tiếp hoặc gián tiếp từ lớp Object
d. Tất cả các câu trên đều sai

26. Lớp nào sau đây dùng để thực hiện các thao tác nhập xuất cơ bản với console

a. System
b. Math
c. String
d. StringBuffer

27. Lớp nào sau đây không phải là lớp bao?

a. String
b. Integer
c. Boolean
d. Character

28. Đoạn mã sau sai chỗ nào?

```
public class Question {
 public static void main(String args[]) {
  Boolean b = new Boolean("TRUE");
  if(b) {
   for(Integer i=0;i<10;++i) {
    System.out.println(i);
   }
  }
 }
}
```

a. Đoạn mã không có lỗi
b. Điều kiện của câu lệnh if phải có kiểu boolean thay vì Boolean
c. Chỉ số của câu lệnh for là int thay vì Integer
d. Câu b) và c) đú

29. Phương thức nào sau đây sẽ làm cho giá trị biến s bị thay đổi

a. s.concat()

b. s.toUpperCase()
c. s.replace()
d. Câu a) và b) đúng

30. Hãy cho biết kết xuất của đoạn chương trình sau:

```
class S1 {
 public static void main(String[] args) {
  new S2();
 }
 S1() {
  System.out.print("S1");
 }
}
class S2 extends S1 {
 S2() {
  System.out.print("S2");
 }
}
```

a. S1
b. S2
c. S1S2
d. S2S1

31. Chọn phát biểu đúng cho hàm khởi tạo
a. Một lớp sẽ kết thừa các hàm khởi tạo từ lớp cha
b. Trình biên dịch sẽ tự động tạo hàm khởi tạo mặc định nếu lớp không định nghĩa hàm khởi tạo
c. Tất cả các hàm khởi tạo có kiểu trả về là void
d. Tất cả các câu trên đều sai

32. Cho biết kết xuất của đoạn chương trình sau:

```
class Question {
 String s = "Outer";
 public static void main(String[] args) {
  S2 s2 = new S2();
  s2.display();
 }
}
class S1 {
 String s = "S1";
 void display() {
  System.out.println(s);
 }
}
class S2 extends S1 {
 String s = "S2";
}
```

a. S1
b. S2
c. null
d. S1S2

33. Một kiểu dữ liệu số có dấu có 2 giá trị +0 và -0 bằng nhau:
a. Đúng
b. Sai
c. Chỉ đúng với kiểu số nguyên
d. Chỉ đúng với kiểu số thực

34. Chọn khai báo tên đúng
a. Big01LongStringWidthMeaninglessName
b. $int
c. bytes
d. Tất cả các câu trên đều đúng

35. Chọn khai báo đúng cho phương thức main()
a. public static void main( )
b. public void main(String[] arg)
c. public static void main(String[] args)
d. public static int main(String[] arg)

36. Chọn thứ tự đúng của các thành phần trong một tập tin nguồn
a. Câu lệnh import, khai báo package, khai báo lớp.
b. Khai báo package đầu tiên; thứ tự của câu lệnh import và khai báo lớp là tùy ý.
c. Khai báo package, câu lệnh import, khai báo lớp
d. Câu lệnh import trước tiên; thứ tự của khai báo package và khai báo lớp là tùy ý.

37. Cho câu lệnh sau:
int[] x = new int[25];
Chọn kết quả đúng sau khi thi hành câu lệnh trên
a. x[24] chưa được định nghĩa
b. x[25] có giá trị 0
c. x[0] = có giá trị null
d. x.length = 25

38. Cho đoạn mã sau:
```
class Q6{
        public static void main(String args[]){
            Holder h = new Holder();
            h.held = 100;
            h.bump(h);
            System.out.println(h.held);
        }
}
class Holder{
            public int held;
            public void bump(Holder theHolder){
```

```
12 :                theHolder.held++;
13 :         }
14 :     }
```

Giá trị in ra của câu lệnh ở dòng thứ 6 là:

a. 0
b. 1
c. 100
d. 101

39. Cho đoạn mã sau:

```
1 : class Q7{
2 :     public static void main(String args[]){
3 :         double d = 12.3;
4 :         Decrementer dec = new Decrementer();
5 :         dec.decrement(d);
6 :         System.out.println(d);
7 :     }
8 : }
9 : class Decrementer{
10 :         public void decrement(double decMe){
11 :                decMe = decMe – 1.0;
12 :         }
13 :     }
```

Giá trị in ra của câu lệnh ở dòng thứ 6 là:

a. 0.0
b. -1.0
c. 12.3
d. 11.3

40. Miền giá trị của biến kiểu short là:

a. Nó phụ thuộc vào nền phần cứng bên dưới
b. Từ 0 đến $2^{16} – 1$
c. Từ -2^{15} đến $2^{15} – 1$
d. Từ -2^{31} đến $2^{31} – 1$

41. Miền giá trị của biến kiểu byte là:
a. Nó phụ thuộc vào nền phần cứng bên dưới
b. Từ 0 đến $2^8 – 1$
c. Từ -2^7 đến $2^7 – 1$
d. Từ -2^{15} đến $2^{15} – 1$

42. Cho biết giá trị của x, a và b sau khi thi hành đoạn mã sau:
1 : int x, a = 6, b = 7;
2 : x = a++ + b++;
a. x = 15, a = 7, b = 8
b. x = 15, a = 6, b = 7
c. x = 13, a = 7, b = 8
d. x = 13, a = 6, b = 7

43. Biểu thức nào sau đây là hợp lệ
a. int x = 6;x = !x;
b. itn x = 6; if (!(x > 3)) {}
c. int x = 6; x = ~x;
d. Câu b) và c) đúng

44. Biểu thức nào sau đây cho x có giá trị dương:
a. int x = -1; x = x >>> 5;
b. int x = -1; x = x >>> 32;
c. byte x = -1; x = x >>> 5;
d. int x = -1; x = x >> 5;

45. Biểu thức nào sau đây hợp lệ
a. String x = “Hello”; int y = 9; x +=y;
b. String x = “Hello”; int y = 9; x = x + y;
c. String x = null; int y = (x != null) && (x.length() > 0) ? x.length() : 0;
d. Tất cả các câu trên đều đúng

46. Đoạn mã nào sau đây in ra màn hình chữ “Equal”:
a.

```
int x = 100; float y = 100.0F;
if (x == y)
{
        System.out.println("Equal");
}
```

b.

```
Integer x = new Integer(100);
Integer y = new Integer(100);
if (x == y)
{
        System.out.println("Equal");
}
```

c.

```
String x = "100"; String y = "100";
if (x == y)
{
        System.out.println("Equal");
}
```

d. Câu a. và c. đúng

47. Cho biết kết quả sau khi thi hành chương trình sau:

```
1 : public class Short{
2 :         public static void main(String[] args){
3 :             StringBuffer s = new StringBuffer("Hello");
4 :             if ((s.length() > 5) &&
5 :                 s.append(" there").equals("False")))
6 :                     ;//do nothing
7 :             System.out.println("value is " + s);
8 :         }
9 : }
```

a. Giá trị xuất là Hello
b. Lỗi biên dịch tại dòng 4 và 5
c. Không có giá trị xuất
d. Thông báo NullPointerException

48. Cho biết kết quả sau khi thực hiện chương trình sau:

```
public class Xor{
        public static void main(String[] args){
            byte b = 10;//00001010
            byte c = 15;//00001111
            b = (byte)(b  ^ c);
            System.out.println(“b contains ” + b);
        }
}
```

a. Kết quả là: b contains 10
b. Kết quả là: b contains 5
c. Kết quả là: b contains 250
d. Kết quả là: b contains 245

49. Cho biết kết quả sau khi biên dịch và thi hành chương trình sau:

```
1 : public class Conditional{
2 :         public static void mai n(String[] args){
3 :             int x = 4;
4 :             System.out.println(“value is ” +
5 :                     ((x > 4 ? 99.99 : 9));
6 :         }
7 : }
```

a. Kết quả là: value is 99.99
b. Kết quả là: value is 9
c. Kết quả là: value is 9.0
d. Lỗi biên dịch tại dòng số 5

50. Cho biết kết quả của đoạn mã sau:

```
int x = 3; int y = 10;
System.out.println(y % x);
```

a. 0
b. 1
c. 2
d. 3

51. Chọn câu khai báo không hợp lệ
a. String s;
b. abstract double d;
c. abstract final double hyperbolCosine();
d. Tất cả các câu trên đều đúng

52. Chọn câu phát biểu đúng
a. Một lớp trừu tượng không thể chứa phương thức final
b. Một lớp final không thể chứa các phương thức trừu tượng
c. Cả a) và b) đều đúng
d. Cả a) và b) đều sai

53. Chọn cách sửa ít nhất để đoạn mã sau biên dịch đúng
```
3 : final class Aaa
4 : {
5 :         int xxx;
6 :         void yyy(){xxx = 1;}
7 : }
8 :
9 :
10 :        class Bbb extends Aaa
11 :        {
12 :            final Aaa finalRef = new Aaa();
13 :
14 :            final void yyy()
15 :            {
16 :                System.out.println("In method yyy()");
17 :                finalRef.xxx = 12345;
18 :            }
19 :        }
```
a. Xóa từ final ở dòng 1
b. Xoá từ final ở dòng 10
c. Xóa từ final ở dòng 1 và 10
d. Không cần phải chỉnh sửa gì

54. Chọn phát biểu đúng cho chương trình sau

```
1 :  class StaticStuff
2 : {
3 :          static int x = 10;
4 :
5 :          static {x += 5;}
6 :
7 :          public static void main(String args[])
8 :          {
9 :              System.out.pritln("x = " + x);
10 :             }
11 :
12 :             static {x /= 5}
13 :         }
```

a. Lỗi biên dịch tại dòng 5 và 12 bỡi vì thiếu tên phương thức và kiểu trả về
b. Chương trình chạy và cho kết quả x = 10
c. Chương trình chạy và cho kết quả x = 15
d. Chương trình chạy và cho kết quả x = 3

55. Chọn phát biểu đúng cho chương trình sau:

```
class HasStatic
{
        private static int x = 100;

        public static void main(String args[])
        {
            HasStatic hs1 = new HasStatic();
            hs1.x++;
            HasStatic hs2 = new HasStatic();
                    hs2.x++;
                    hs1 = new HasStatic();
                    hs1.x++;
                    HasStatic.x++;
                    System.out.println("x = " + x);
```

```
                }
        }
```

a. Chương trình chạy và cho kết quả x = 102
b. Chương trình chạy và cho kết quả x = 103
c. Chương trình chạy và cho kết quả x = 104
d. Tất cả các câu trên đều sai

56. Cho đoạn mã sau:

```
1 : class SuperDuper
2 : {
3 :         void aMethod(){}
4 : }
5 :
6 : class Sub extends SuperDuper
7 : {
8 :         void aMethod(){}
9 : }
```

Hãy chọn từ khóa chỉ phạm vi hợp lệ đứng trước aMethod() ở dòng 8

a. default
b. protected
c. public
d. Tất cả các câu trên đều đúng

Ø Đoạn mã sau dùng cho 2 câu hỏi tiếp theo

```
package abcde;

public class Bird{
        protected static int referneceCount = 0;
        public Bird(){referenceCount++;}
        protected void fly(){…}
        static int getRefCount(){return referenceCount;}
}
```

57. Chọn phát biểu đúng cho lớp Bird trên và lớp Parrot sau:

```
1 :  package abcde;
2 :
3 :  class Parrot extends abcde.Bird{
4 :          public void fly(){
5 :              //
6 :          }
7 :          public int getRefCount(){
8 :              return referenceCount;
9 :          }
10 :         }
```

a. Lỗi biên dịch ở dòng 4 tập tin Parrot.java vì phương thức fly() là protected trong lớp cha và lớp Bird và Parrot nằm trong cùng package

b. Lỗi biên dịch ở dòng 4 tập tin Parrot.java vì phương thức fly() là protected trong lớp cha và public trong lớp con.

c. Lỗi biên dịch ở dòng 7 tập tin Parrot.java vì phương thức getRefCount() là static trong lớp cha.

d. Chương trình biên dịch thành công nhưng sẽ phát sinh Exception khi chạy nếu phương thức fly() của lớp Parrot không được gọi

58. Chọn phát biểu đúng cho lớp Bird trên và lớp Nightingale sau:

```
package singers;

class Nightingale extends abcde.Bird{
        Nightingale(){ refernceCount++;}

        public static void main(String args[]){
            System.out.print(“Before: “ + refernceCount);
            Nightingale florence = new Nightingale();
            System.out.print(“After: “ + refernceCount);
                florence.fly();
            }
        }
```

a. Kết quả trên màn hình là: Before: 0 After: 2
b. Kết quả trên màn hình là: Before: 0 After: 1
c. Lỗi biên dịch ở dòng 4 của lớp Nightingale vì không thể overidde thành viên static
d. Lỗi biên dịch ở dòng 10 của lớp Nightingale vì phương thức fly() là protected trong lớp cha.

59. Chọn phát biểu đúng
a. Chỉ kiểu dữ liệu cơ sở mới được chuyển đổi kiểu tự động; để chuyển đổi kiểu dữ liệu của biến tham chiếu phải sử dụng phép ép kiểu
b. Chỉ biến tham chiếu mới được chuyển đổi kiểu tự động; để chuyển kiểu của 1 biến kiểu cơ sở phải sử dụng phép toán ép kiểu
c. Cả kiểu dữ liệu cơ sở và kiểu tham chiếu đều có thể chuyển đổi tự động và ép kiểu
d. Phép ép kiểu đối với dữ liệu số có thể cần phép kiểm tra khi thực thi

60. Dòng lệnh nào sau đây sẽ không thể biên dịch:
1 : byte b = 5;
2 : char c = ‘5’;
3 : short s = 55;
4 : int i = 555;
5 : float f = 555.5f;
6 : b = s;
7 : i = c;
8 : if (f > b)
9 :   f = i;
a. Dòng 3
b. Dòng 4
c. Dòng 5
d. Dòng 6

61. Chọn dòng phát sinh lỗi khi biên dịch

```
1 :  byte b = 2;
2 :  byte b1 = 3;
3 :  b = b * b1;
```

a. Dòng 1
b. Dòng 2
c. Dòng 3
d. Tất cả các câu trên đều đúng

62. Trong đoạn mã sau kiểu dữ liệu của biến result có thể là những kiểu nào?

```
1 :  byte b = 11;
2 :  short s =13;
3 :  result  = b * ++s;
```

a. byte, short, int, long, float, double
b. boolean, byte, short, char, int, long, float, double
c. byte, short, char, int, long, float, double
d. int, long, float, double

63. Cho đoạn chương trình sau:

```
1 :  class Cruncher{
2 :      void crunch(int i){
3 :              System.out.println("int version"):
4 :      }
5 :      void crunch(String s){
6 :       System.out.println("String version");
7 :      }
8 :
9 :      public static void main(String[] args){
10 :             Cruncher crun = new Cruncher();
11 :             char ch = 'p';
12 :             crun.crunch(ch);
13 :      }
14 :   }
```

a. Dòng 5 sẽ không biên dịch vì phương thức trả về kiểu void không thể overridde

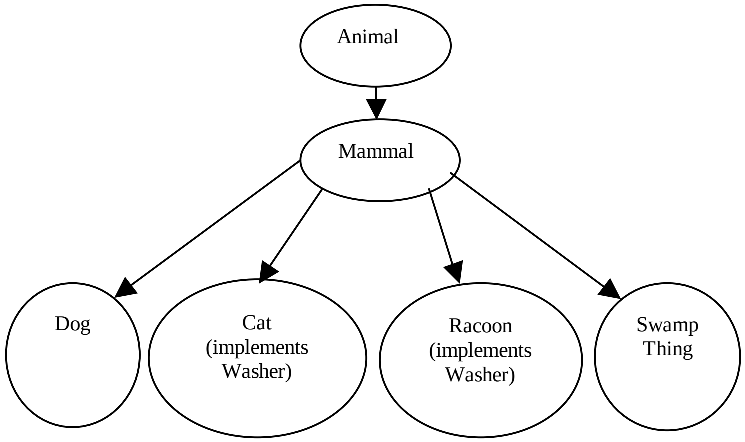
b. Dòng 12 sẽ không biên dịch vì không có phiên bản nào của phương thức crunch() nhận vào tham số kiểu char
c. Đoạn mã biên dịch được nhưng sẽ phát sinh Exception ở dòng 12
d. Chương trình chạy và in ra kết quả: int version

64. Chọn phát biểu đúng
a. Tham chiếu của đối tượng có thể được chuyển đổi trong phép gán nhưng không thể thực hiện trong phép gọi phương thức
b. Tham chiếu của đối tượng có thể được ép kiểu trong phép gọi phương thức nhưng không thể thực hiện trong phép gán
c. Tham chiếu của đối tượng có thể được chuyển đổi trong phép gọi phương thức và phép gán nhưng tuân theo những quy tắc khác nhau
d. Tham chiếu của đối tượng có thể được chuyển đổi trong phép gọi phương thức và phép gán và tuân theo những quy tắc giống nhau

65. Cho đoạn mã như bên dưới. Hãy cho biết dòng nào không thể biên dịch

```
1 :   Object ob = new Object();
2 :   String stringarr[] = new String[50];
3 :   Float floater = new Float(3.14f);
4 :   ob = stringarr;
5 :   ob = stringarr[5];
6 :   floater = ob;
7 :   ob = floater;
```

a. Dòng 4
b. Dòng 5
a. Dòng 6
b. Dòng 7



Hình sau áp dụng cho các câu 66, 67, 68

66. Cho đoạn mã sau:

1 :Dog rover, fido;
2 :Animal anim;
3 :
4 :rover = new Dog();
5 :anim = rover;
6 :fido = (Dog)anim;

Hãy chọn phát biểu đúng

a. Dòng 5 không thể biên dịch
b. Dòng 6 không thể biên dịch
c. Đoạn mã biên dịch thành công nhưng sẽ phát sinh Exception tại dòng 6
d. Đoạn mã biên dịch thành công và có thể thi hành

67. Cho đoạn mã sau:

1 :Cat sunflower;
2 :Washer wawa;
3 :SwampThing pogo;
4 :
5 :sunflower = new Cat();

```
6 :wawa = sunflower;
7 :pogo = (SwampThing)wawa;
```

Hãy chọn phát biểu đú

a. Dòng 6 không thể biên dịch; cần có một phép ép kiểu để chuyển từ kiểu Cat sang kiểu Washer

b. Dòng 7 không thể biên dịch vì không thể ép từ kiểu interface sang kiểu class

c. Đoạn mã sẽ dịch và chạy nhưng phép ép kiểu ở dòng 7 là thừa và có thể bỏ di

d. Đoạn mã biên dịch thành công nhưng sẽ phát sinh Exceptiono ở dòng 7 vì kiểu lớp của đối tượng trong biến wawa lúc thi hành không thể chuyển sang kiểu SwampThing

68. Cho đoạn mã sau

```
1 :Racoon rocky;
2 :SwampThing pogo;
3 :Washer w;
4 :
5 :rocky = new Racooon();
6 :w = rocky;
7 :pogo = w;
```

a. Dòng 6 sẽ không biên dịch; cần phải có phép ép kiểu để chuyển từ kiểu Racoon sang kiểu Washer

b. Dòng 7 sẽ không biên dịch; cần có phép ép kiểu để chuyển từ kiểu Washer sang kiểu SwampThing

c. Đoạn mã sẽ biên dịch nhưng sẽ phát sinh Exception ở dòng 7 vì chuyển đổi kiểu khi thực thi từ interface sang class là không được phép

d. Đoạn mã sẽ biên dịch và sẽ phát sinh Exception ở dòng 7 vì kiểu lớp của w tại thời điểm thực thi không thể chuyển sang kiểu SwampThing

69. Cho đoạn mã sau:

```
1 :  for (int i = 0; i < 2; i++){
2 :           for (int j = 0; j < 3; j++){
```

```
            if (i == j){
                    continue;
            }
            System.out.println("i = " + i + "j = " + j);
        }
}
```

Dòng nào sẽ là một trong số các kết quả được in ra?

a. i = 0 j = 0
b. i = 2 j = 1
c. i = 0 j = 2
d. i = 1 j = 1

70. Cho đoạn mã sau:

```
outer:  for (int i = 0; i < 2; i++){
        for (int j =0; j < 3; j++){
            if (i == j){
                    continue outer;
            }
            System.out.println("i = " + i + "j = " + j);
        }
}
```

Dòng nào sẽ là một trong số các kết quả được in ra?

a. i = 0 j = 0
b. i = 0 j = 1
c. i = 0 j = 2
d. i = 1 j = 0

71. Chọn vòng lặp đúng

a.

```
while (int i < 7){
        i++;
        System.out.println("i is " + i);
}
```

b.

```
int i = 3;
while (i){
        System.out.println("i is " + i);
}
```

c.

```
int j = 0;
for(int k = 0; j + k != 10; j++, k++){
        System.out.println("j is " + j + " k is " + k);
}
```

d.

```
int j = 0;
do{
        System.out.println("j is " + j++);
        if (j == 3) { continue loop;}
}while (j < 10);
```

72. Cho biết kết xuất của đoạn mã sau

```
int x = 0, y = 4, z  = 5;
if ( x > 2){
        if (y < 5){
                System.out.println("message one");
        }
        else{
                System.out.println("message two");
        }
}
    else if (z > 5){
        System.out.println("message three");
    }
        else{
                System.out.println("message four");
        }
```

a. message one
b. message two
c. message three

d. message four

73. Chọn phát biểu đúng cho đoạn mã sau:

```
1 :  int j = 2;
2 :  switch (j){
3 :          case 2:
4 :                  System.out.println("value is two");
5 :          case 2 + 1:
6 :                  System.out.println("value is three");
7 :                  break;
8 :          default:
9 :                  System.out.println("value is" + j);
10 :                         break;
11 :         }
```

a. Đoạn mã không hợp lệ bỡi biểu thức ở dòng 5
b. Biến j trong cấu trúc switch() có thể là một trong các kiểu: byte, short, int hoặc long
c. Kết xuất của chương trình chỉ là dòng: value is two
d. Kết xuất của chương trình chỉ là dòng: value is two và value is three

74. Cho biết kết quả sau khi dịch và thực thi đoạn chương trình sau:

```
import java.awt.*;

public class Test extends Frame {
  Test() {
          setSize(300,300);
          Button b = new Button("Apply");
          add(b);
  }

    public static void main(String args[]) {
          Test  f = new Test();
    f.setVisible(true);
```

13.   }
14. }

a) Có lỗi biên dịch tại dòng 11 bởi vì constructor ở dòng 4 không khai báo public.

b) Chương trình biên dịch thành công nhưng có sẽ ném ra exception khi thực thi câu lệnh ở dòng thứ 7.

c) Chương trình hiển thị frame trống.

d) Chương trình hiển thị 1 nút nhấn (Button) sử dụng font chữ mặc định cho nhãn của Button. Button chỉ đủ lớn để bao quanh nhãn của nó.

e) Chương trình hiển thị nút nhấn (Button) dùng font chữ mặc định cho nhãn nút. Nút nhấn sẽ choán tất cả vùng hiển thị của frame.

75. Nếu 1 frame dùng bộ quản lý trình bày (layout manager) là GridLayout và không chứa bất kỳ panel hay container nào khác bên trong nó thì tất cả những components khi đưa vào trong frame này có cùng kích thước như nhau (ngang, dọc)?

a) Đúng.
b) Sai.

76. Nếu 1 frame dùng bộ quản lý trình bày (layout manager) mặc định và không chứa bất kỳ panel nào bên trong thì tất cả những components bên trong frame là cùng kích thước (ngang, dọc) ?

a) Đúng.
b) Sai.

77. Với bộ quản lý trình bày BorderLayout không nhất thiết các vùng phải có chứa các components.

a) Đúng.
b) Sai.

78. Bộ quản lý trình bày mặc định cho 1 khung chứa kiểu Panel là:

a) FlowLayout
b) BorderLayout
c) GridLayout
d) GridBagLayout

79. Một Container có bộ quản lý trình bày là GridBagLayout thì mỗi component sẽ có kích thước bằng nhau khi thêm vào khung chứa (container) này?
a) Đúng
b) Sai

80. Bạn có thể tạo ra cửa sổ chính của ứng dụng bằng cách gọi:
Frame f = new Frame("Main Frame");
Nhưng khi bạn chạy chương trình thì Frame không hiển thị. Dòng nào bên dưới sẽ làm hiển thị Frame.
a) f.setSize(300, 200);
b) f.setBounds(10, 10, 500, 400);
c) f.setForeground(Color.white);
d) f.setVisible(true);

81. Đối tượng nào bên dưới có thể chứa 1 menubar (chọn những câu đúng)
a) Panel
b) ScrollPane
c) Frame
d) Menu

82. Sau khi tạo 1 frame bằng câu lệnh *Frame f = new Frame()* và tạo menu bar bằng câu lệnh *MenuBar mb = new MenuBar()*, làm thế nào để gắn *MenuBar* tên *mb* vào *f*.
a) f.add(mb)
b) f.setMenu(mb)
c) f.addMenu(mb)
d) f.setMenuBar(mb)

Phụ Lục B: Đáp án trắc nghiệm kiến thức

1.c; 2.b; 3.d; 4.c; 5.d; 6.d; 7.c; 8.a; 9.d; 10.a; 11.a; 12.c; 13.b; 14.d; 15.a; 16.d; 17.a; 18.c; 19.a; 20.c; 21.a; 22.a; 23.a; 24.a; 25.c; 26.a; 27.a; 28.d; 29.d; 30.c; 31.b; 32.a; 33.b; 34.d; 35.c; 36.c; 37.d; 38.d; 39.c; 40.c; 41.c; 42.c; 43.d; 44.a; 45.d; 46.d; 47.a; 48.b; 49.c; 50.b; 51.d; 52.b; 53.a; 54.d; 55.c; 56.d; 57.c; 58.a; 59.c; 60.d; 61.c; 62.d; 63.d; 64.d; 65.c; 66.d; 67.d; 68.b; 69.c; 70.d; 71.c; 72.d; 73.d; 74.e; 75.a; 76.b; 77.a; 78.a; 79.b; 80.d; 81.c; 82.d